Roger Wagner Publishing

Presents

the

Interrupt Experimenter's Kit

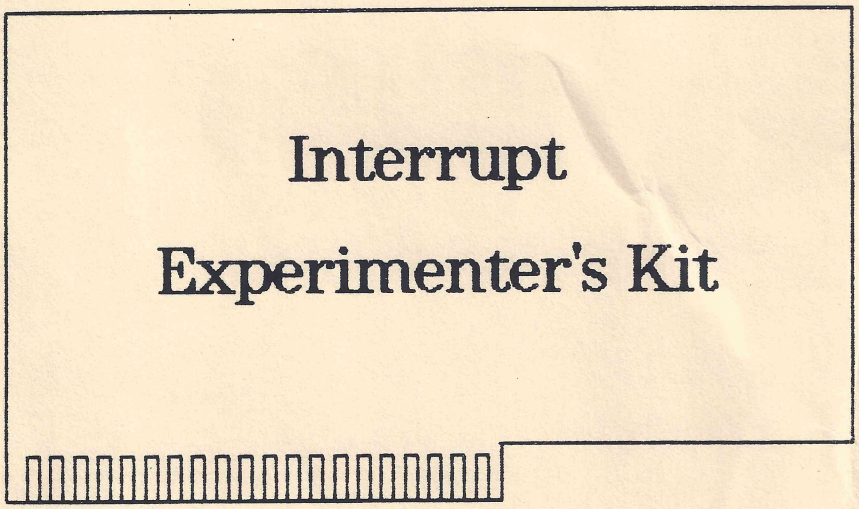
10761 Woodside Ave. - Suite "E"
Santee, CA 92071

Roger Wagner Publishing

Presents

the

Interrupt Experimenter's Kit



10761 Woodside Ave. - Suite "E"
Santee, CA 92071

Then Apple's...

DOS 3.3 Standard is a copyrighted program of Apple Computer, Inc. licensed to Roger Wagner Publishing, Inc. to distribute for use only in combination with The Interrupt Experimenter's Kit.

APPLE COMPUTER, INC. MAKES NO WARRANTIES, EITHER EXPRESS OR IMPLIED, REGARDING THE ENCLOSED SOFTWARE PACKAGE, ITS MERCHANTABILITY OR ITS FITNESS FOR ANY PARTICULAR PURPOSE. THE EXCLUSION OF IMPLIED WARRANTIES IS NOT PERMITTED BY SOME STATES. THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. THIS WARRANTY PROVIDES YOU WITH SPECIFIC LEGAL RIGHTS. THERE MAY BE OTHER RIGHTS THAT YOU MAY HAVE WHICH VARY FROM STATE TO STATE.

And now on with our program!

First, our legal stuff...

ROGER WAGNER PUBLISHING, INC.
CUSTOMER LICENSE AGREEMENT

IMPORTANT: The Roger Wagner Publishing, Inc. software product that you have just received from Roger Wagner Publishing, Inc., or one of its authorized dealers, is provided to you subject to the Terms and Conditions of this Software Customer License Agreement. Should you decide that you cannot accept these Terms and Conditions, then you must return your product with all documentation and this License marked "REFUSED" within the 30 day examination period following the receipt of the product.

1. License. Roger Wagner Publishing, Inc. hereby grants you upon your receipt of this product, a nonexclusive license to use the enclosed Roger Wagner Publishing, Inc. product subject to the terms and restrictions set forth in this License Agreement.

2. Copyright. This software product, and its documentation, is copyrighted by Roger Wagner Publishing, Inc. You may not copy or otherwise reproduce the product or any part of it except as expressly permitted in this License.

3. Restrictions on Use and Transfer. The original and any backup copies of this product are intended for your personal use in connection with a single computer. You may not distribute copies of, or any part of, this product without the express written permission of Roger Wagner Publishing, Inc.

LIMITATION ON WARRANTIES AND LIABILITY

ROGER WAGNER PUBLISHING, INC. AND THE PROGRAM AUTHOR SHALL HAVE NO LIABILITY OR RESPONSIBILITY TO PURCHASER OR ANY OTHER PERSON OR ENTITY WITH RESPECT TO ANY LIABILITY, LOSS OR DAMAGE CAUSED OR ALLEGED TO BE CAUSED DIRECTLY OR INDIRECTLY BY THIS SOFTWARE, INCLUDING, BUT NOT LIMITED TO ANY INTERRUPTION OF SERVICE, LOSS OF BUSINESS OR ANTICIPATORY PROFITS OR CONSEQUENTIAL DAMAGES RESULTING FROM THE USE OR OPERATION OF THIS SOFTWARE. SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF IMPLIED WARRANTIES OR LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU.

*** THE INTERRUPT EXPERIMENTER'S KIT ***

An Introduction to Using Interrupts
on the Apple II family of computers.

by Dan Fischer, Morgan P. Caffrey
and Roger Wagner

TABLE OF CONTENTS

THE INTERRUPT EXPERIMENTERS´S KIT

By Dan Fischer, Morgan P. Caffrey and Roger Wagner

INTRODUCTION

Computers are dumb, unimaginative, critters. They do as they are told, no more, no less. They follow a list of instructions, from the first to the last, without deviation. Of course the program instruction itself may change the sequence but that is just part of following instructions.

However, as with everything else in this inconsistent world, there are exceptions. Resets and power failures are traumatic events in the life of an executing program. They interrupt the flow of the program. Actually they don´t interrupt it as much as they terminate it. They interrupt a program much as a speeding truck might interrupt the life of a lazy skunk. The interrupt never ends.

"Interrupt" implies a temporary condition. There are ways to interrupt a program´s flow, without the program´s "knowledge" or control, and without interfering with the logical process of the program. These interrupts require hardware however, and software smart enough to recognize the situation and handle it appropriately.

There are two hardware signals within the APPLE which cause the Apple´s microprocessor, hereinafter referred to as the "6502," to save its place in one program and go off and execute another program. This interrupt service program can return control to the original program is such a way that the original program won´t even know that it has been interrupted.

Suppose you are talking to someone on the phone when the doorbell rings. You excuse yourself from the phone, take care of the person at the door, and return to the phone to finish telling your neighbor about the latest sale of minis and micros. If you handled the door interrupt well you will pick up your story where you left off and your listener will follow right along without any "What was that?"s or "Would you start over?"

If, however, the person at the door happens to be the Grim Reaper, here to take you to the Promised Land, you will never return to your ´interrupted´ conversation.

Resets and power failures are like the Grim Reaper, programs never return .

When the APPLE is prepared to handle interrupts, it is prepared to run two separate programs at once (at least in outward appearance). Let's refer to the original program, the one that gets interrupted, as the 'foreground' task and to the program that gets run by the interrupt as the 'background' task. From the user's standpoint both programs are running at the same time. From the 6502's standpoint it always executes instructions one after the other, never two at once.

Although it is almost commonplace to say it, microprocessors are FAST. Running two programs at once, performing very simple to very complex functions, can happen without the user even being aware of the slightest slowdown. In the Apple II it is possible to run a 300 character-per-second printer (listing a text file from disk) at the exact same time you are entering data into your word processor.

The foreground task doesn't "know" about the background task. It doesn't have to consider it at all. It may freely treat the computer as its private resource (although space must be reserved for the background program to exist).

On the other hand, the background task must be polite. It must consider the foreground task and not do anything to interfere with the foreground task. It must be more polite than the environmentalist in the forest. You know, the one who takes only pictures and leaves only footprints. The background program must not leave footprints. Leaving even the slightest footprint on a computer program has a tendency to scramble your screen designs and do odd kinds of rounding to your numbers.

I can hear you asking "If it's all so complicated why even do foreground/background stuff at all? Why not just let well enough alone and let the APPLE do its thing with a single program?" You can do that. You usually do. But there are times when you could set your APPLE to doing a task for you and it probably has time left over to do another task.

USING A CLOCK

For example, if you have a clock card in your APPLE you might want to display the time on the screen and update

the time every second or every minute. It would be a terrible waste of the APPLE's capability to use it for a clock to the exclusion of anything else.

Using interrupts you can set the background task to updating the screen clock and leave the foreground task to do the work at hand. The amount of time used each second to perform such a task is a mere fraction of one percent of the available processing time each second.

PRINTING and PROGRAMMING

Or maybe you get bored waiting for your printer to finish printing. In a way so does your APPLE. It is waiting for your printer about 96% of the time when it is printing. That's time you could be using your APPLE for something else if you had a way for the printer to tell the APPLE when it is ready for another character.

One solution for this problem is to buy a buffer card (a smart bucket of RAM, Random Access Memory, cells) and have the hardware quickly gulp characters destined for the printer and feed them out one at a time to the printer. This can be an expensive proposition and can't handle documents much larger than the bucket. When the bucket fills you wait anyway, although not for as long overall as you would have without it.

Another method is to dedicate a small portion of your computer's memory to the task of handling the printer in background mode. Much less expensive. And it can be made to work at many more problems than just printing.

Now let's get a little technical. There are two types of interrupts in the APPLE. They are called IRQ and NMI. The NMI is the "non-maskable interrupt". You probably guessed that the IRQ is a "maskable" interrupt.

Whenever a non-maskable interrupt occurs the 6502 will process it. Nothing will keep it from doing so. The presence of the physical interrupting signal cannot be masked off from the 6502. The RESET key works in a similar way. When you press RESET, the Apple must stop whatever it is doing to process the RESET command.

The APPLE is constructed in such a way that there are times when it performs tasks that are time-dependent. In feeding data to the disk drive the 6502 is used as a timer. Its cyles are like the ticks (or tocks, if you prefer) of a software clock.

If a non-maskable interrupt were to occur while you were writing a sector to the disk the diskette's data integrity would be destroyed. The NMI is sufficiently dangerous that we don't recommend using it.

The IRQ, however, is a quite manageable beast. IRQ stands for Interrupt ReQuest. When a hardware device senses a condition for which it is supposed to generate an interrupt it pulls the line called IRQ low (lowers the voltage). The 6502, if its program has told it to honor interrupt requests, responds by saving its place in the program it is executing and begins to execute the background interrupt program. When the interrupt routine is finished executing it points the 6502 back to where it was before it was interrupted and returns control back to the foreground task.

HOW AN INTERRUPT IS PROCESSED

"How does it do all this?" you ask. Since interrupt service routines are invariably written in machine language, all future discussions will assume you know a least a little about assembly language programming. It's not required that you be an expert in this regard, but you should know the difference between a JSR and JMP!

If you don't have a background in this area, you might consider reading a book like "ASSEMBLY LINES: THE BOOK", Published by Roger Wagner Publishing as a good start to learning assembly language programming. Also, to create any of your own interrupt routines you'll need a good assembler such as the MERLIN assembler, also published by Roger Wagner Publishing.

CLI, SEI, and RTI

An interrupt causes what can be thought of as a "conditional JSR". It is conditional because the computer has the option of ignoring interrupt requests.

As mentioned earlier, there are times, such as while writing to the disk, that an interrupt would produce very undesireable results. Therefore, two assembly language commands are provided to allow, or prevent, interrupt requests from being recognized. These are the CLI and SEI commands.

When you first turn on the Apple, one of the first machine language instructions executed is a SEI instruction. This stands for "Stop External Interrupts", and tells the 6502 to ignore any future interrupt requests. (Remember, RESET is not a request, it's a demand!).

If you want to have the 6502 honor an interrupt request, then the instruction CLI (for Clear Interrupt mask) must be used. Once this is executed, the next interrupt request to come along will be honored.

This is where the 'JSR' part of the 'conditional JSR' concept comes in. When an interrupt request does come along, the 6502 will do something very much like a JSR to a routine of your chosing at that point. This is also why interrupt routines must be written with a certain amount of care. Because the JSR-like operation can occur at almost <u>any time</u>, your interrupt routine must restore any registers or other important memory locations it has altered before it returns to the normally running foreground program.

What this usually means is that you must deliberately save the Accumulator and X and Y registers, and then restore them when your routine is ready to return to the foreground program.

How is the pseudo-JSR done? Because it can be triggered at almost any location in your foreground program, there is no equivalent command to the JSR that you're already familiar with. Instead, the hardware that generates the interrupt works hand in hand with the 6502 to break program execution away from the foreground program to go to the background interrupt routine.

Once your interrupt routine has control, no future interrupt requests can "re-interrupt" things until you return to the foreground program. When <u>you</u> are ready to return, you use the instruction RTI (for Return from Interrupt) to return to the foreground program.

WHERE DO INTERRUPTS THEMSELVES COME FROM?

So far we've given a general picture of how interrupts are handled, but where do IRQ's, interrupt requests, come from in the first place?

An interrupt is generated by changing the voltage on the Apple's 'bus'. The bus is a term that refers to the

collection of wires that run through the system that carry signals telling the 6502 what address is being accessed and other interesting information.

These lines are most visible as the slots into which peripheral cards are plugged. Each connector on a peripheral card attaches to a wire that makes up the lines of the bus. Two of these wires are the NMI and IRQ lines. Ordinarily, nothing is hooked up to either of these lines.

That's where The Interrupt Experimenter's Kit comes in. The Apple as shipped doesn't have anything provided to generate interrupts, and that's why so few people are familiar with how to use them. When you plug the circuit card that's included in The Interrupt Experimenter's Kit (the Interrupt Source Card) into slot #4 of your Apple, you are attaching some special circuitry which will generate interrupts on the Apple's IRQ line, which will then allow you to experiment with your own interrupt routines.

The question now becomes, how do you turn the interrupts generated by the card on and off?

The card that is provided with this package produces interrupts at a regular rate of about 250 interrupts per second. The card is activated by accessing a special memory location in the Apple. It happens that the Apple is set up so that each peripheral slot has 16 bytes of memory set aside for it in the general area of $C080 to $C0FF. To determine which block of memory is available to a given slot, you just add the slot number to '8' in the address. For example, for slot #4, the memory range is $C0(8+4)0 to $C0(8+4)F, or specifically, $C0C0 to $C0CF. Any BASIC or machine language instruction (such as PEEK, POKE, LDA, STA or BIT) that references any address in the range of $C0C0 to $C0CF will turn on the interrupts on the card.

Remember, these are interrupt requests (IRQ's), and nothing will happen unless a CLI instruction is also executed at some point.

To turn off the card, you must access the address space allocated for the card itself (256 bytes this time), which is $C(N)00 through $C(N)FF. Thus for slot #4, accessing any address from $C400 to $C4FF would turn off the card. Generally, you should also remember to execute an SEI to disable interrupts, but this may depend on your application.

The card can also be turned off by hitting RESET, but this does not perform the SEI to disable interrupts at the 6502 level, so keep this in mind. Also remember that DOS automatically does an SEI to disable interrupts when either reading or writing to the disk, but it will restore the existing condition (interrupts enabled or disabled) when it's done. You'll notice this in any of the demos in this package if you CATALOG a disk while a demo is running. The demo will stop during the CATALOG, but resume when the disk turns off.

WHAT HAPPENS DURING AN INTERRUPT

Now that we know the Apple at least has the capacity to handle interrupts, and we've provided a way to generate them, let's look at what's necessary to set up an interrupt routine, and what actually happens when an interrupt comes along.

THE INTERRUPT VECTOR

Before letting interrupts loose in the system, the first thing we have to do is to tell the Apple what to do with them when they happen. This means we've got to tell the computer where our interrupt handling routine will be located in memory.

This is done by means of the 'interrupt vector'. A vector is simply a pair of bytes somewhere in the computer that holds the address of some other location. In this case, the vector we're concerned with are the bytes $3FE and $3FF. These two bytes point at where the 6502 should go to when an interrupt comes along.

If we wanted to put our own routine starting at location $300 for example, we would put a '$00' at $3FE and a '$03' at $3FF.

Once this is done, (and our routine is loaded at $300), we can safely enable interrupts with the CLI command, and then turn on the interrupt card. Assuming this much has happened, let's see what happens when the first interrupt occurs.

When an interrupt occurs, the first thing that happens is that the 6502 saves the address of where it's currently at in the foreground program. This is so that it will be able to resume where it left off when the interrupt routine is finished. It does this by pushing the values in something called the Program Counter (also called the PC register) onto the stack. The Program Counter is a register inside the 6502, sort of like the Accumulator, except that it holds two bytes that point to where in a given program the 6502 is currently executing.

After the Program Counter is saved, the Status Register is also pushed on the stack. Remember, the Status Register holds the results of comparisons, the overflow flag, the carry bit, and all sorts of other important information, so it's easy to see why this also gets saved right away. The IRQ bit in the Status Register is also set here to prevent further interrupts from effecting things until the end of the given interrupt routine, at which point interrupts will be re-enabled when the Status Register is restored.

Once the most important things have been saved, the 6502 then jumps to the address pointed to by the last two bytes of memory, $FFFE and FFFF. On most Apple II+, //E and //c machines, this will point to $FA40. This is the Monitor's interrupt routine, where the real software handling of the interrupt starts.

Let's look at the code at $FA40 to see what happens here. Go into the Monitor by typing CALL -151 from Applesoft. Then type in FA40L. You should get:

```
FA40-  85 45      STA $45        ; save Acc.
FA42-  68         PLA            ; retrieve Status Reg.
FA43-  48         PHA            ; copy Status to stack
FA44-  0A         ASL
FA45-  0A         ASL
FA46-  0A         ASL            ; shift int flg to N
FA47-  30 03      BMI $FA4C      ; branch if BRK
FA49-  6C FE 03   JMP ($3FE)     ; JMP via vector
```

The comments on the right are shown here to help explain what's going on, and won't be displayed on your screen.

Since this routine is about to use the Accumulator to do some testing, it first saves the contents of the Accumulator in location $45. This is important to you because you will have to use the contents of $45 later

at the end of your routine to restore the Accumulator before doing the RTI to restore everything properly.

The next thing that happens is that the Status Register is pulled off the stack and then recopied back onto it. The net effect here is that the stack is unchanged, but there is now a copy of the Status Register in the Acuumulator. This is because the routine wants to determine whether an interrupt or a BRK instruction has occurred. The BRK instruction is handled by this same part of the Monitor. The distinction is made by examining bit 4 of the Status Register. If it's set, a BRK (Break) instruction has occurred. If clear, it's an interrupt.

It takes three ASL's (Arithmetic Shift Left) to put bit 4 into the N (negative/positive) flag of the Status Register, at which point the BMI is used to test it. If a BRK has happened, it branches to the BREAK routine at $FA4C. If (as we would hope) it is an interrupt, it jumps to the part of memory pointed to by the bytes at $3FE and $3FF. (Aren't you glad you set this up ahead of time? Imagine where it might go if $3FE,3FF were random values!)

It is at this point that your interrupt routine gains control.

You might hink that the most trivial interrupt routine would be just a single RTI. This is factually true, but does not work when implemented. That's because you have to remember to restore the Accumulator before the RTI. Thus, the simplest working routine is:

```
$0300-  A5 45      LDA $45          ; restore Acc.
$0302-  40         RTI              ; return to program
```

With the Accumulator restored, when the RTI is executed it first restores the Status Register by pulling a value back off of the stack. This also has the effect of re-enabling interrupts (remember CLI clears bit 3 of the Status Register to enable interrupts). Next the Program Counter is restored with another two pulls. It then returns to the foreground program that was running when the interrupt first occurred.

Another way of looking at the complete process is to say that when an interrupt occurs, the equivalent of the following occurs:

```
STA $45          ; Save Accumulator.
LDA PC           ; Not possible, but the idea of it.
PHA              ; Put the low byte of PC on stack.
LDA PC+1
PHA              ; Put high byte of PC on stack.

PHP              ; Put Status Register on stack.
SEI              ; Protect from further interrupts.
JMP ($3FE)       ; Jump to user's routine.
```

When your program is finished, the RTI performs the equivalent of this:

```
PLP              ; Restore Status Register (= CLI)
                 ; You've already restored at least
                   the Acc. haven't you?
RTS              ; Return to foreground program.
                   (pulls PC,PC+1 off stack to do it)
```

For those who are really deep thinking, it should also be obvious (because the system wouldn't work otherwise) that interupts are really excluded from the very beginning of the PC and Status Register save operations. That is to say, the 6502's interrupt handling during the first few cycles can't be reinterrupted.

THE FIRST EXPERIMENT

Now that we've covered the theory, let's try out your first interrupt experiment. Since we only need to change a few locations in memory, an Applesoft BASIC program will be a good way to set everything up.

If you haven't done so already, turn off the power on your Apple, and install the Interrupt Source Card in slot #4 of your computer. Then replace the cover and re-boot on the Interrupt Experimenter's Kit diskette.

You'll see a title page, and if you press a key, the disk will CATALOG and you'll get the Applesoft prompt (]). When you get the Applesoft prompt, type in NEW and then enter this program:

```
 0 REM INTERRUPT TEST PROGRAM
 5 PRINT CHR$(7): REM NORMAL BELL SOUND
10 POKE 768,165: POKE 769,69: REM LDA $45 INSTR.
20 POKE 770,64: REM RTI INSTRUCTION
30 POKE 771,88: POKE 772,76: REM CLI, RTS INSTR.
40 POKE 1014,0: POKE 1015,3: REM ($3FE,3FF) = $300
50 CALL 771: REM ENABLE INTERRUPTS
```


1022 96 1023

```
60 POKE 49999,0: REM ACCESS $C0C0 TO TURN ON CARD
70 PRINT CHR$(7): REM INT BELL SOUND
```

When you RUN this program, you should first hear a beep, and then something that's more like a buzz than a beep.

Let's look at the listing. Line 5 prints the Apple's 'beep' character, Control-G. This is for comparison to the later sound.

Lines 10 and 20 POKE into memory, starting at location $300, the values for the following machine language program:

```
0300-  A5 45        LDA $45
0302-  40           RTI
```

This iwill be our trivial interrupt routine. Line 30 creates a short program that will enable interrupts. It would disassemble like this:

```
0303-  58           CLI
0304-  60           RTS
```

Line 40 sets the interrupt vector at $3FE,3FF to point to location $300, which is the start of our interrupt routine.

To enable interrupts, i.e. tell the 6502 to honor interrupt requests, line 50 makes a CALL to the simple instructions at $303 to clear the interrupt mask.

To start the interrupts, line 60 accesses location $C0C0 to turn on the Interrupt Source Card.

So, why does the second Control-G sound funny? It's because interrupts are now enabled and being processed 250 times per second by our routine at $300. This means that the speaker doesn't get cliked as often during the Control-G beep sound, and thus has a lower pitched sound.

Congratulations! You've entered and run your first interrupt driven program. From here on out, all that happens is to figure out what kind of things can be put in the interrupt routine to do fun things!

This demo program is an easy way to test your system to make sure interrupts can be used and are working properly. If the program does not behave as described

above, read on through the next section and make sure the interrupt vector ($3FE,3FF) and the program at $300 have been set up properly. Also double-check your Applesoft program listing for typing errors. Other possible problems include putting the Interrupt Source Card in a slot other than slot #4, or broken wires or components on the card itself.

A tested copy of the above Applesoft program is included on your Interrupt Experimenter's Kit diskette under the name APPLESOFT INT TEST PROGRAM. Also, if you have the Doubletime Printer F8 ROM installed in your computer, you must either delete the LDA $45 instruction from the interrupt routine, or use the Universal Interrupt Return listed in Appendix B of this manual.

OTHER INTERRUPT CONSIDERATIONS

After you've RUN the test program above, hit RESET to turn off the interrupt card. Then enter the Monitor with a CALL -151. Now type in:

```
3FE 3FF
```

and press RETURN. The screen should print out:

```
03FE-  00 03
```

This confirms that the interrupt vector has been set up properly. Now type in:

```
300L
```

The top of the screen should now look like this:

```
0300-  A5 45      LDA $45
0302-  40         RTI
0303-  58         CLI
0304-  60         RTS
```

As mentioned earlier, the two instructions at 300 and 302 are the actual interrupt routine; 303 and 304 are the instructions to enable interrupts before turning on the card.

PROBLEMS WITH INTERRUPTS

Sad to say, there are problems with interrupts on the Apple, and there aren't any really easy answers to them.

The first is that interrupts are disabled whenever DOS accesses a diskette. The main drawback to this is that an interrupt driven software clock will lose time whenever the disk is used. This is not a problem if you are timing something within a program where the disk is not accessed, but does prevent you from timestamping your DOS file accurately with a software clock.

NOTE: Hardware clocks do not suffer from this problem because they have an on-board chip which is the clock, as opposed to a software counter driven by the 6502 and interrupts. Many clock cards do have a provision for enabling interrupts though so that you can monitor the clock within other background programs.

The second problem is more serious, and really complicates a number of operations. That problem is the use of location $45 by the Monitor interrupt handling routine. Remember how the interrupt handler stored the contents of the Accumulator in location $45?

Imagine for a moment that the foreground program is using location $45 at the instant the interrupt occurs. Unknown to the foreground program, the contents of location $45 are destroyed by the interrupt handler.

Are there routines in the computer that use location $45? Yes. Some important ones are the IOSAVE and IOREST routines at $FF4A and $FF3F in the Monitor. IOSAVE stores the A, X, Y, P and S registers in locations $45 through $49. IOREST restores all these registers using the contents of $45 through $49.

Where you get into trouble is when a foreground program uses IOSAVE at the beginning to save all the registers, does some operation, and then uses IOREST to restore all of the registers. If an interrupt occurs during the foreground routine after IOSAVE has been called, location $45 is destroyed by the interrupt. When IOREST is called at the end of the foreground program, the Accumulator will not hold the original value. This leads to unpredictable, and usually fatal (for the program), results.

What about programs that don't use IOSAVE and IOREST? Unfortunately, there are a few other things that use location $45, most notably DOS! That's right.

Whenever many DOS commands are processed, location $45 is used. In fact, whenever a BSAVE or BLOAD is done that uses the address or length parameters (such as in

BSAVE FILE, A$2000, L$2000), the routine that does the math also uses location $45.

Although DOS disables interrupts when it actually accesses the disk, during the time the calculations are done, it is unprotected. If an interrupt occurs during this period, the calculated result can be incorrect, again leading to unpredictable results, such as mis-saved file lengths, etc.

FOR THIS REASON, YOU SHOULD NEVER LOAD OR SAVE FILES UNDER DOS 3.3 WITH INTERRUPTS ENABLED AND ACTIVE IN THE COMPUTER!

How do clock cards and modems that use interrupts avoid this problem? They don't really. They caution their users not to allow interrupts to be enabled while accessing disks. The risk of interupt damage to location $45 is a direct function of the rate of the interrupts.

This is easy to understand. If interrupts are being generated only once every minute, then in terms of a percentage, very little of the actual foreground program is interrupted. Many clock cards only generate an interrupt once every second, and this is still only 1 out of 1,000,000 cyles for the computer. However, as the interrupt rate increases (and some clock cards support rates of up to 10,000 interrupts per second), the odds of an interrupt occurring in many different parts of a given routine increase.

On the "other side of the interrupt", any interrupt program that uses the I/O hooks, i.e. either prints through COUT or uses KEYIN, can also get into trouble.

This is because DOS uses location $45 rather freely, and DOS is activated for every character that goes in or out through the usual I/O channels. Thus, there is the potential for conflict for each character printed or input. DOS also uses location $45 during a CATALOG printout, during an INIT and during the opening and closing of files.

If you do want to print something through COUT during your interrupt routine, there is the option of changing the vectors to COUT (CSW + $36,$37) within the interrupt routine before the printing is done, and then restoring them before returning with the final RTI. See the CLOCK demo on the Interrupt Experimenter's Kit and the discussion later in this manual for more information on this.

There are some other considerations when using interrupts also. For example, reading the paddles is a time-based operation. The value that the paddle read routine in the monitor returns to Applesoft (or a machine language program) depends on the time it takes for a certain capacitor in the computer to charge up.

If you are stealing time with your interrupt routine, paddle values will come back that are artificially high to the foreground program. Of course, since your background program is immune to further interrupts, you can read the paddles accurately from within it.

On the Apple //e computer, interrupts are disabled whenever the screen (40 or 80 columns) is scrolled. This means that a software clock would also lose time whenever the screen scrolled.

If all this sounds very discouraging, take heart! Most of this can be managed as long as you are careful to remember where the pitfalls lie. That's why it's important to be aware of them.

For example, if you want to read the paddles during an Applesoft program that uses interrupts, just turn off interrupts while you're reading the paddle. We already mentioned that interrupt routines that want to print something can temporarily keep DOS out of the picture while they are working.

You'll probably find that even with the limitations mentioned, there are a number of fun things you can do with interrupts in your own programs.

A "SHELL" INTERRUPT ROUTINE

To continue our experiments, let's look at the "shell" of a simple interrupt routine. In fact, this shell is so simple it doesn't do anthing except set up a bunch of NOP's as the routine!

On the following two pages is the listing for this interrupt routine. Let's examine each line of the source listing to see exactly how it works.

The first thing to notice is that this routine sets up its own interrupt vector. When you think about it, this is a good approach. The routine itself is only directly called once, and this is when the setup is done. After that, successive calls are done indirectly as each

interrupt initiates a jump to the functional part of the routine via the interrupt vector.

Specifically, lines 12 through 15 in the source listing load the low and high order bytes of the address of START, and put these in $3FE,$3FF to set up the interrupt vector. Line 16 then clears the interrupt flag to enable the 6502 to recognize interrupts, and line 17 turns on the Interrupt Source Card. The RTS on line 18 (DONE) then returns to the calling program (or user).

Lines 20 through 22 are a data storage area that will be used to store the contents of the A, X and Y registers in just a moment.

When the first interrupt occurs, the 6502 will jump to START via the interrupt vector. Lines 24 through 26 store the contents of the A, X and Y registers. Remember, this is done because when our interrupt routine is finished, we have to return to the foreground program with everything in the same condition as when we entered the interrupt routine. Although the Accumulator has presumably already been saved in location $45 by the Monitor's interrupt handler (and in fact is no longer the "true" value), this form of the source listing will be easier to alter when alternate Monitor ROM's are discussed in Appendix B.

Lines 28 through 30 are just some symbolic NOP's that represent where you would put your own program if you were writing an interrupt routine.

When the routine is finished and ready to return to the foreground program, lines 32 through 34 restore the A, X and Y registers before the final RTI. Again, since location $45 holds the true Accumulator value, SAVREG is not used to restore the Accumulator.

```
                        1     ********************************
                        2     * "SHELL" INTERRUPT ROUTINE    *
                        3     ********************************
                        4
                        5              OBJ   $300
                        6              ORG   $300
                        7
                        8     IRQ      EQU   $3FE        ; INTERRUPT VECTOR
                        9     INTCRD   EQU   $C0C0       ; SOFTSWITCH TO TURN
                                                            ON CARD
                       10                                 ; $C0(8+N)0
                       11
0300: A9 12             12     INIT     LDA   #<START
0302: 8D FE 03          13              STA   IRQ
0305: A9 03             14              LDA   #>START
0307: 8D FF 03          15              STA   IRQ+1       ; SET UP IRQ VECTOR
030A: 58                16              CLI               ; ENABLE INTERRUPTS
030B: AD C0 C0          17              LDA   INTCRD      ; TURN ON CARD
030E: 60                18     DONE     RTS
                       19
030F: 00                20     SAVREGS  HEX   00          ; ACC
0310: 00                21              HEX   00          ; X-REG
0311: 00                22              HEX   00          ; Y-REG
                       23
0312: 8D 0F 03          24     START    STA   SAVREGS
0315: 8E 10 03          25              STX   SAVREGS+1
0318: 8C 11 03          26              STY   SAVREGS+2
                       27
031B: EA                28              NOP
031C: EA                29              NOP               ; YOUR PROGRAM HERE
031D: EA                30              NOP
                       31
031E: A5 45             32     RETURN   LDA   $45         ; RESTORE ACC.
0320: AE 10 03          33              LDX   SAVREGS+1   ; RESTORE X
0323: AC 11 03          34              LDY   SAVREGS+2   ; RESTORE Y
                       35
0326: 40                36     FINISH   RTI
```

THE CLOCK SIMULATOR

This next interrupt routine actually does something. It simulates a clock function by using the interrupts as a timer. Each time an interrupt occurs, the routine increments certain counters and displays an elapsed time in the upper left hand corner of the screen.

Before describing the actual lines in the routine, let's take an overview of how the routine is expected to perform.

Once the routine is loaded, the interrupt vectors set up and interrupts started, the routine will be called approximately 250 times per second.

If the interrupts occurred just once each second, then we could just increment a second-counter register to keep track of the elapsed time. In this case though we'll need another counter to count each interrupt, so as to be able to increment the time counter once every 250 calls to the routine.

On each 250th call then, we'll click the speaker to make a "tick" noise (alternate "tock" is left as an exercise to the reader!..) and print the time (in hex) on the screen.

To see how this general plan is implemented, look at the listing on the following pages.

The initialization (INIT) on lines 21 through 34 are patterned after the SHELL INTERRUPT ROUTINE discussed earlier, but you'll notice the time data register TIME is initialized to zero for the clock counter and the cycle counter (INTCNT for Interrupt Counter) is set to '250'.

When an interrupt occurs, lines 47 through 57 save A, X and Y, but also save the horizontal screen cursor position (CH), BASL,H and a temporary register called YSAV1.

These last special bytes are all used by the print routine that will be called within the interrupt routine. Because we must restore the complete system before returning from the interrupt routine, any bytes altered must be saved so they can be restored later.

TEST (lines 59-62) decrements the interrupt counter, and at the same time checks to see if it has reached zero

yet. If not, the routine jumps to the exit section, FINSEC, which will be discussed in just a moment.

When INTCNT reaches zero, the program falls through "TEST" to NEWSEC on line 64. This resets the interrupt counter back to 250 and increments the TIME counter.

CLICK then executes a short loop that clicks the speaker four times. Between each click is a short wait so that the click loop is slow enough to be heard. (Without a wait the loop would be too fast for the speaker to even respond to!).

PRINT (lines 81 through 98) first does the equivalent of an HTAB 1: VTAB 1. We could just do the printing now, but because DOS is constantly processing all output, printing with DOS active during an interrupt routine can change data elsewhere in the computer (most importantly, zero page addresses) that are impossible to keep track of from the interrupt routine.

An easy way of avoiding this is to take DOS out of the output path by pointing COUT directly at the screen output routine at $FDF0. This keeps DOS out of trouble for a while. Lines 92-94 use the PRNTAX routine in the Monitor to print the time in hexidecimal. Lines 95-98 then restore the COUT vector (CSW = $36,37) to its original value.

This brings us to FINSEC, which is pretty much the usual exit routine. Lines 100-111 restore all the registers and zero page bytes that may have been changed by the interrupt routine.

Note that the label FREQ is set equal to an assumed number of interrupts per second. Because the actual frequency of individual Interrupt Source Cards vary widely, you may want to change to value of FREQ to give a more accurate time for your own card.

```
                        1    ********************************
                        2    *       CLOCK SIMULATOR        *
                        3    ********************************
                        4
                        5             OBJ   $300
                        6             ORG   $300
                        7
                        8    IRQ      EQU   $3FE        ; INTERRUPT VECTOR
                        9    SPKR     EQU   $C030
                        10   INTCRD   EQU   $C0C0       ; SOFTSWITCH TO TURN
                                                           ON CARD
                        11   CH       EQU   $24         ; HORIZ CURSOR
                                                           POSITION
                        12   BASL     EQU   $28
                        13   VTAB     EQU   $FC22
                        14   PRNTAX   EQU   $F941
                        15   YSAV1    EQU   $35         ; USED BY COUT
                                                           (PRNTAX)
                        16   WAIT     EQU   $FCA8
                        17   CHROUT   EQU   $FDF0       ; VIDEO OUT
                        18   CSW      EQU   $36         ; OUTPUT VECTOR
                        19   FREQ     EQU   250         ; INT SOURCE CARD
                                                           FREQ.
                        20
0300: A9 2C             21   INIT     LDA   #<START
0302: 8D FE 03          22            STA   IRQ
0305: A9 03             23            LDA   #>START
0307: 8D FF 03          24            STA   IRQ+1       ; SET UP IRQ VECTOR
030A: A9 00             25            LDA   #$00
030C: 8D 21 03          26            STA   TIME        ; SET TIME = 0
030F: 8D 22 03          27            STA   TIME+1
0312: A9 2C             28            LDA   #<FREQ      ; # OF INTERRUPTS
                                                           PER SECOND
0314: 8D 23 03          29            STA   INTCNT      ; SET FOR COUNT-DOWN
0317: A9 01             30            LDA   #>FREQ
0319: 8D 24 03          31            STA   INTCNT+1
031C: 58                32            CLI               ; ENABLE INTERRUPTS
031D: AD C0 C0          33            LDA   INTCRD      ; TURN ON CARD
0320: 60                34   DONE     RTS
                        35
0321: 00 00             36   TIME     HEX   0000        ; TIME COUNTER
0323: 00 00             37   INTCNT   HEX   0000        ; CURRENT INTERRUPT
                                                           OF SERIES
                        38
0325: 00                39   SAVEREG  HEX   00          ; ACC STORAGE LOC.
0326: 00                40            HEX   00          ; X-REG
0327: 00                41            HEX   00          ; Y-REG
0328: 00                42            HEX   00          ; CH
0329: 00                43            HEX   00          ; BASL
032A: 00                44            HEX   00          ; BASL+1
032B: 00                45            HEX   00          ; YSAV1
```

```
                        46
032C: 8D 25 03  47  START   STA  SAVEREG     ; SAVE ACC
032F: 8E 26 03  48          STX  SAVEREG+1   ; SAVE X
0332: 8C 27 03  49          STY  SAVEREG+2   ; SAVE Y
0335: A5 24     50          LDA  CH
0337: 8D 28 03  51          STA  SAVEREG+3   ; SAVE CH
033A: A5 28     52          LDA  BASL
033C: 8D 29 03  53          STA  SAVEREG+4
033F: A5 29     54          LDA  BASL+1
0341: 8D 2A 03  55          STA  SAVEREG+5   ; SAVE BASL,BASH
0344: A5 35     56          LDA  YSAV1
0346: 8D 2B 03  57          STA  SAVEREG+6   ; SAVE YSAV1
                        58
0349: CE 23 03  59  TEST    DEC  INTCNT      ; ARE WE IN THE
                                               MIDDLE OF A
                                               SEGMENT?
034C: D0 4B     60          BNE  FINSEC      ; BRANCH IF YES (NO
                                               ACTION)
034E: CE 24 03  61          DEC  INTCNT+1
0351: 10 46     62          BPL  FINSEC
                        63
0353: A9 2C     64  NEWSEC  LDA  #<FREQ      ; REFRESH COUNTER
0355: 8D 23 03  65          STA  INTCNT
0358: A9 01     66          LDA  #>FREQ
035A: 8D 24 03  67          STA  INTCNT+1
035D: EE 21 03  68          INC  TIME        ; TIME = TIME + 1
0360: D0 03     69          BNE  CLICK
0362: EE 22 03  70          INC  TIME+1
                        71
0365: A0 04     72  CLICK   LDY  #$04       ; # OF TIMES TO
                                               CLICK SPEAKER
0367: 2C 30 C0  73  C1      BIT  SPKR        ; CLICK THE SPEAKER
036A: 88        74          DEY
036B: F0 08     75          BEQ  PRINT
036D: A9 0A     76          LDA  #$0A        ; TIME TO WAIT
036F: 20 A8 FC  77          JSR  WAIT        ; WAIT TO CLICK
                                               AGAIN
0372: 18        78          CLC
0373: 90 F2     79          BCC  C1          ; ALWAYS
                        80
0375: A9 00     81  PRINT   LDA  #00
0377: 85 24     82          STA  CH          ; HTAB 1
0379: 20 24 FC  83          JSR  VTAB+2      ; VTAB 1
037C: A5 36     84          LDA  CSW
037E: 48        85          PHA
037F: A5 37     86          LDA  CSW+1
0381: 48        87          PHA              ; SAVE CURRENT CSW
                                               VECTOR
0382: A9 F0     88          LDA  #<CHROUT
0384: 85 36     89          STA  CSW
0386: A9 FD     90          LDA  #>CHROUT
```

```
0388: 85 37      91             STA  CSW+1       ; CSW = $FDF0
038A: AE 21 03   92             LDX  TIME
038D: AD 22 03   93             LDA  TIME+1
0390: 20 41 F9   94             JSR  PRNTAX      ; PRINT TIME IN HEX
                                                   SECONDS
0393: 68         95             PLA
0394: 85 37      96             STA  CSW+1
0396: 68         97             PLA
0397: 85 36      98             STA  CSW         ; RESTORE COUT HOOKS
                 99
0399: AD 28 03   100  FINSEC    LDA  SAVEREG+3
039C: 85 24      101            STA  CH          ; RESTORE CH
039E: AD 29 03   102            LDA  SAVEREG+4
03A1: 85 28      103            STA  BASL
03A3: AD 2A 03   104            LDA  SAVEREG+5
03A6: 85 29      105            STA  BASL+1      ; RESTORE BASL
03A8: AD 2B 03   106            LDA  SAVEREG+6
03AB: 85 35      107            STA  YSAV1       ; RESTORE YSAV1
03AD: AE 26 03   108            LDX  SAVEREG+1   ; RESTORE X
03B0: AC 27 03   109            LDY  SAVEREG+2   ; RESTORE Y
                 110
03B3: A5 45      111  RETURN    LDA  $45         ; RESTORE ACC
03B5: 40         112  FINISH    RTI
```

USING INTERRUPT ROUTINES FROM APPLESOFT

It is possible to use interrupt routines from Applesoft BASIC, as demonstrated by the first experiment in this manual. To save having to use POKE's to set up things though, some extra routines have been included in the package to make using interrupts a little bit easier. These routines are IRQ ON.TB and IRQ OFF.TB.

USING THE ROUTINES WITH THE WORKBENCH

Included with your Interrupt Experimenter's Kit is a free Trial Size Toolkit diskette. This is sample of a series of other Roger Wagner Publishing products called "The Toolbox Series".

The Toolbox system is a special utility called The Workbench plus a large set of new commands that can be added to any Applesoft program. These commands are added (usually in less than a minute) using the Workbench utility.

The easiest way to use the routines IRQ ON.TB and IRQ OFF.TB is to use the Workbench utility that is included in the Trial Size Toolbox package. If you do not want to do this, skip to the next section on using the routines without the Workbench utility.

If you are using the Workbench, the Toolbox files IRQ ON.TB and IRQ OFF.TB can be added just like any other Toolbox command. The functions of these commands are as follows:

IRQ ON.TB

This command will set up the interrupt vector at $3FE,3FF to the address of your choice, enable interrupts with the required CLI instruction, and turn on the Interrupt Source Card (assumed to be in slot #4).

IRQ OFF.TB

This command disables interrupts by setting the interrupt mask flag with an SEI instruction and turns off the Interrupt Source Card (still assumed to be in slot #4).

Here is a sample Applesoft program that uses the SHELL INTERRUPT ROUTINE as an example:

```
 1  CALL PEEK(175) + 256 * PEEK(176) - 46
 5  D$ = CHR$(4) : REM CONTROL-D DOS COMMANDS
20  PRINT D$;"BLOAD SHELL INTERRUPT ROUTINE, A768":
    REM A$300

100 PRINT CHR$(7): REM PRINT NORMAL BELL SOUND
110 &"IRQON",768: REM TURN ON INTERRUPTS
120 PRINT CHR$(7): REM PRINT INT BELL SOUND
130 &"IRQOFF": REM TURN OFF INTERRUPTS
140 PRINT CHR$(7): REM PRINT NORMAL BELL AGAIN
150 END
```

The idea behind this program is to load the SHELL INTERRUPT routine into memory at address 768 ($300 hex).

The syntax for turning on interrupts is to use the IRQ ON.TB command, followed by a comma and the address that you want the interrupt vector to point to. This would usually be the first byte of your interrupt routine. In this case that is location 768.

When this program is run, the BELL sound (Control-G) should first be normal, then sound like a buzz, and then return to normal.

The syntax for turning off the interrupts is to just use the command IRQ OFF.TB. No address variables are required.

IRQ ON.TB and IRQ OFF.TB can be used as many times as you like within your Applesoft program to turn interrupts off and on. Also note that you can use a hexidecimal address in IRQ ON.TB by just putting a dollar sign in front of the hex number you wish to use as the address. For example, line 110 of the sample program above could also look like this:

```
110 &"IRQON",$300: REM TURN ON INTERRUPTS
```

SUMMARY OF IRQ ON.TB AND IRQ OFF.TB

IRQ ON.TB: Length: 76 bytes ($4B)

Syntax:
```
&"NAME" [,aexpr|,hexnum]
&"NAME" [,ADDRESS]
```

Sample:
```
&"IRQON"
&"IRQON",AD
&"IRQON",$300
```

Function: IRQ ON.TB enables interrupts with the CLI instruction and turns on the Interrupt Source Card in slot #4. If an address is specified, the interrupt vector at $3FE,3FF will be set to that address.

Limitations: IRQ ON.TB assumes the Interrupt Source Card is in slot #4. If you have the card in another slot, either re-assemble the source file with line #77 changed accordingly, or alter byte 72 of the object file to 128 + 16 * slot # (Example: for slot #3, byte 72 = 128 + 16 * 3 = 176). Also, if a hex value is used for the address, it must be specified as shown in the example, and not as a string variable (i.e. don't use quotes or a string variable).

IRQ OFF.TB: Length: 5 bytes ($5)

Syntax: &"NAME"

Sample: &"IRQOFF"

Function: IRQ OFF.TB disables interrupts with the SEI instruction and turns off the Interrupt Source Card in slot #4.

Limitations: IRQ OFF.TB assumes the Interrupt Source Card is in slot #4. If you have the card in another slot, either re-assemble the source file with line #20 changed accordingly, or alter byte 2 of the object file to 192 + slot # (Example: for slot #3, byte 2 = 192 + 3 = 195).

USING THE ROUTINES WITHOUT THE WORKBENCH

If you do not want to use the Workbench to add the commands IRQ ON.TB and IRQ OFF.TB to your Applesoft program, then just follow the steps here.

IRQ ON.TB

This routine will set up the interrupt vector at $3FE,3FF to the address of your choice, enable interrupts with the required CLI instruction, and turn on the Interrupt Source Card (assumed to be in slot #4).

IRQ OFF.TB

This routine disables interrupts by setting the interrupt mask flag with an SEI instruction and turns

off the Interrupt Source Card (still assumed to be in slot #4).

Here's an Applesoft BASIC program that uses these two routines along with the SHELL INTERRUPT ROUTINE as an example.

```
 5  D$ = CHR$(4) : REM CONTROL-D DOS COMMANDS
10  PRINT D$;"BLOAD IRQ ON.TB, A768": REM A$300
20  PRINT D$;"BLOAD IRQ OFF.TB, A848": REM A$350
30  PRINT D$;"BLOAD SHELL INTERRUPT ROUTINE, A864":
    REM A$360

100 PRINT CHR$(7): REM PRINT NORMAL BELL SOUND
110 CALL 768,864: REM TURN ON INTERRUPTS
120 PRINT CHR$(7): REM PRINT INT BELL SOUND
130 CALL 848: REM TURN OFF INTERRUPTS
140 PRINT CHR$(7): REM PRINT NORMAL BELL AGAIN
150 END
```

The idea behind this program is to first load the IRQ ON.TB and IRQ OFF.TB routines into memory, along with the SHELL INTERRUPT ROUTINE.

The syntax for turning on interrupts is to call the IRQ ON.TB routine, followed by a comma and the address that you want the interrupt vector to point to. This would usually be the first byte of your interrupt routine. In this case that would be location 864 ($360 hex).

When this program is run, the BELL sound (Control-G) should first be normal, then sound like a buzz, and then return to normal.

The syntax for turning off the interrupts is to just call the address of IRQ OFF.TB. No address variables are required.

IRQ ON.TB and IRQ OFF.TB can be called as many times as you like within your Applesoft program to turn interrupts off and on. Also note that you can use a hexidecimal address in IRQ ON.TB by just putting a dollar sign in front of the hex number you wish to use as the address. For example, line 110 of the sample program above could also look like this:

```
110 CALL 768,$360: REM TURN ON INTERRUPTS
```

```
                        1     **************************
                        2     *                        *
                        3     *   IRQ VECTOR SETUP     *
                        4     * BY OSMERIOD & PENTALIC *
                        5     *       8/1/84           *
                        6     *                        *
                        7     **************************
                        8     *
                        9     *
                        10    ****************************
                        11    * NOTE: SYNTAX FROM BASIC IS:
                        12    * &"NAME",$FFFF OR 65536
                        13    ****************************
                        14    *
                        15              OBJ   $300
                        16    *
                        17    A2L       EQU   $3E
                        18    A2H       EQU   $3F
                        19    CHRGET    EQU   $B1
                        20    CHRGOT    EQU   $B7
                        21    GETNUM    EQU   $FFA7
                        22    LINNUM    EQU   $50
                        23    FRMNUM    EQU   $DD67
                        24    GETADR    EQU   $E752
                        25    IRQ       EQU   $3FE
                        26    *
                        27
                        28    * CHECK NEXT CHARACTER
                        29    * AND BRANCH IF NO PARAMETERS
                        30    * ARE GIVEN TO IRQON
                        31    * ELSE CHECK IF PARAMETER IS
                        32    * HEX OR DEC & BRANCH ACCORDINGLY
                        33
8000: 20 B7 00          34    ENTRY     JSR   CHRGOT
8003: F0 42             35              BEQ   IRQON
8005: C9 24             36              CMP   #'$         ; HEX VALUE?
8007: D0 2E             37              BNE   FPNUM       ; NO
                        38    *
8009: A0 00             39              LDY   #$00
800B: 20 B1 00          40    XFER      JSR   CHRGET
800E: C9 30             41              CMP   #'0
8010: 90 14             42              BCC   PROCESS
8012: C9 3A             43              CMP   #':
8014: 90 08             44              BCC   X1
8016: C9 41             45              CMP   #'A
8018: 90 0C             46              BCC   PROCESS
801A: C9 47             47              CMP   #'G
801C: B0 08             48              BCS   PROCESS
                        49    *
801E: 09 80             50    X1        ORA   #$80
8020: 99 00 02          51              STA   $200,Y
```

```
8023: C8          52            INY
8024: D0 E5       53            BNE  XFER
                  54   *
8026: A0 00       55   PROCESS  LDY  #$00
8028: 20 A7 FF    56            JSR  GETNUM
802B: A5 3E       57            LDA  A2L
802D: 85 50       58            STA  LINNUM
802F: A5 3F       59            LDA  A2H
8031: 85 51       60            STA  LINNUM+1
8033: A0 00       61            LDY  #$00
8035: F0 06       62            BEQ  HOOK
                  63   *
8037: 20 67 DD    64   FPNUM    JSR  FRMNUM
803A: 20 52 E7    65            JSR  GETADR
                  66   *
803D: A5 50       67   HOOK     LDA  LINNUM
803F: 8D FE 03    68            STA  IRQ
8042: A5 51       69            LDA  LINNUM+1
8044: 8D FF 03    70            STA  IRQ+1
                  71   *
                  72   * FIND END OF STATEMENT OR LINE
                  73   * AND SET TXTPTR. ACTIVATE THE
                  74   * CARD AND ENABLE INTERRUPTS
                  75   * (CARD ASSUMED TO BE IN SLOT 4)
                  76
8047: AD C0 C0    77   IRQON    LDA  $C0C0      ; $C0(8+N)0
804A: 58          78            CLI
804B: 60          79            RTS

                   1   ****************************
                   2   *                          *
                   3   *      IRQ TURN OFF        *
                   4   * BY OSMERIOD & PENTALIC   *
                   5   *        8/1/84            *
                   6   *                          *
                   7   ****************************
                   8   *
                   9   ******************************
                  10   * NOTE: SYNTAX FROM BASIC IS:
                  11   * &"NAME"
                  12   ******************************
                  13            OBJ  $300
                  14
                  15   IRQ       EQU  $3FE
                  16
                  17   * DEACTIVATE CARD (ASSUMES SLOT
                  18   * FOUR) AND TURN OFF IRQ'S
                  19
8000: AD 00 C4    20   IRQOFF   LDA  $C400      ; TURN OFF CARD
8003: 78          21            SEI             ; DISABLE INTS
8004: 60          22   DONE     RTS
```

USING INTERRUPTS WITHOUT THE INTERRUPT SOURCE CARD

The Interrupt Source Card is provided in the Interrupt Experimenter's Kit so that you can have an easy and immediate way of generating interrupts in your own Apple.

Does this mean that the programs you write that use interrupts will only work on your Apple? Not necessarily. There are many peripheral devices for the Apple that can generate their own interrupts. The most common examples are clock cards, serial interface cards and modems. There are also combination devices like the Prometheus VERSACARD that can generate interrupts too.

To see how these can generate interrupts, refer to the manuals for the device you have (or have access to), and see what is required to turn the interrupts on. The only difference in using interrupts from a peripheral card other than the Interrupt Source Card is that a location other than $C080 may have to be accessed to turn the card on. Otherwise all the programs on the Interrupt Experimenter's Kit diskette will work fine.

With a little thought, you should be able to think of all sorts of possible applications for interrupts. What might these things be? Here's a short list.

1. Display the time (you need a real hardware clock for this).
2. Run the Printer in background mode (See the note about Doubletime Printer later in this manual).
3. Upload a text file through a modem to a distant computer.
4. Download data from a distant computer to a RAM or disk buffer (each character arrives through the modem).
5. Reading scientific or industrial measurement equipment (from Apple's game port or other sources).
6. Hi-Res game background features.
7. Using the computer to monitor time and control BSR household controllers while still being able to run foreground programs.

Remember, in adding one or more of these functions you would not be dedicating your computer, you would be sharing the resource among various tasks.

OTHER HARDWARE CONSIDERATIONS

SHARED ROM

The Apple's $C800-CFFF memory is "shared". Each peripheral slot selects to use it if it needs it and ignores it if it doesn't. If a peripheral card has ROM on it, the card is required to select the C800 space which removes entirely, without a trace, the previous selection. For one device to use the address space it must turn off all devices.

If the interrupt routine uses a device that uses $C800 space (such as downloading text material from a hard-disk or sending a character to an 80-column display) the interrupt routine must take care to restore the $C800 user who may have been in charge when the interrupt occured. Apple Computer has defined a protocol for use of $C800 space. Unfortunately, not everyone (including Apple) has always implemented this protocol.

The protocol requires any user of the $C800 space to inform the general system by leaving the slot number in location $7F8 in the form of "Cs", where s=slot number. Reading Cs00 should turn on the ROM for the board in slot s. Reading CFFF should turn off all ROMS. Some, or all, of this protocol has been ignored by some or all versions of the APPLE COM CARD, VIDEX VIDEOTERM, AXLON RAMDISK, SORRENTO VALLEY ASSOCIATES APPLECACHE. There are probably other boards that do not follow the convention.

PAGED RAM CARDS

Remember we said that interrupt programs shouldn't even leave footprints? Interrupt programs must not tiptoe through the RAM CARDS. Most RAM cards on the market do not permit users to read the status of the read/write or bank select circuitry. This means that if the interrupt program changes the RAM card status it will not be able to restore it to what the foreground task expects. This nearly guarantees a program crash. The only exceptions to this oversight that we know of are the Microsoft 16K card and the Prometheus Expand-A-Ram 128k card.

Another, more dangerous, potential problem can occur with RAM cards. Suppose the foreground task is using a RAM card for data storage. What would happen if there were an interrupt when the RAM card was enabled for

read? Well, the 6502 would look to locations FFFE and FFFF for the vector to the IRQ routine. There's an old principle in computers, "Data executes like garbage!" Such a situation would probably send the APPLE off into the weeds.

DEMONSTRATION PROGRAMS AND OTHER GOODIES...

The remainder of this manual consists of Appendices that provide additional information about interrupts and your Apple.

Appendix A lists the known conflicts between the Monitor and DOS (i.e. location $45) and the use of interrupts and will help you anticipate possible problems using interrupts on your Apple.

Appendix B is a description of how the Monitor ROM can be altered to avoid these conflicts, and includes a special offer for a print spooling software package (Doubletime Printer) that includes a modified F8 Monitor ROM that not only fixes the interrupt problem (location $45 conflict) for Apple II computers (sorry, not for the //e or //c), but also fixes the problem with the automatic converting of lower case letters when you copy over them with the cursor!

There is also a description of a "Universal Interrupt Return" procedure that will let your interrupt routines work with either the standard or modified F8 Monitor ROM described in this section.

Appendix C contains explanations of some additional demonstration programs included on the Interrupt Experimenter's Kit diskette.

Well, if you heed the cautions mentioned throughout this manual, and use some care in writing your interrupt routines, you should be pretty safe in using interrupts. Interrupts can be a very interesting area of computer programming. We hope this package has been of help in teaching you about this fascinating part of your Apple computer. Gotta go now...It looks like someone is a about to push the RESET butto_____

APPENDIX A: LOCATION $45 CONFLICTS

As mentioned earlier in the manual, there are a number of routines built into the Apple Monitor and DOS that use location $45. The problem created here is that if an interrupt occurs while any of these routines are being used, the calculations they perform or data that they handle will be destroyed.

Following is a list of the known locations in the Monitor and DOS that reference location $45, along with comments about the usage. Applesoft does not use location $45.

This list is provided to give you an idea of where possible conflicts may arise when using interrupts within your own programs.

Although the Monitor can be modified to avoid the location $45 usage problem, modifying DOS is a more substantial project. Information on modifying the Monitor is provided in the next Appendix. Short of these techniques, the best solution is brought to mind by the story of the man who goes into the doctor and tells him "I hurts when I do this {performing a certain unusual motion with his body}". Replied the doctor, "Well, then don't do that!"

MONITOR REFERENCES TO LOCATION $45
(Auto-Boot ROM)

Actual Location of Reference: $FA40 (STA $45)
Part of the Routine: IRQ ($FA40)
Purpose/Comments: Jumped to by vector at $FFFE, FFFF on any BRK, or on IRQ if bit 3 of Status Register is clear (i.e. CLI to enable interrupts).

Actual Location of Reference: $FF42 (LDA $45)
Part of the Routine: IOREST ($FF3F) {also called RESTORE}
Purpose/Comments: Used to restore Accumulator and other registers. See the Monitor 'G' (for Go) routine. IOREST is also used by other programmers writing their own routines, and this use is difficult, if not impossible to document.

Actual Location of Reference: $FF4A (STA $45)
Part of the Routine: IOSAVE ($FF4A) {also called SAVE}
Purpose/Comments: Used on to save A, X, Y, P and S registers. Like IOREST, IOSAVE is also used by other programmers writing their own routines, and thus its use is difficult to document.

Actual Location of Reference: $FEB9 (JSR $FF3F)
Part of the Routine: GO ($FEB6)
Purpose/Comments: Used to set A, X, Y, P and S registers and then jump to an address specified by the user (Example: 300G). Most people don't even know about this feature of the Monitor so it's not likely to be a problem.

DOS 3.3 REFERENCES TO LOCATION $45

Actual Location of Reference: $A157 (LDA $45)
Part of the Routine: SETDFLTS ($A0D1)
Purpose/Comments: Sets slot, drive, volume, etc. defaults on any DOS command.

Actual Location of Reference: $A200 (ROL $45)
Part of the Routine: GETNUM ($A1B9)
Purpose/Comments: Converts ASCII string numbers (hex or decimal) into actual integer bytes for DOS. Used during any numeric calculation of drive, slot, volume, length, address, etc.

Actual Location of Reference: $A2CB (LDA $45)
Part of the Routine: CMDHDLR ($A2A8)
Purpose/Comments: Part of command handler for DOS. File manager setup used by various DOS commands.

Actual Location of Reference: $A77E (STA $45)
Part of the Routine: LOCBUF ($A764)
Purpose/Comments: Finds free file buffer for DOS file data transfer.

Actual Location of Reference: $ADB9 (STX $45)
Part of the Routine: CATHNDLR ($AD98)
Purpose/Comments: Part of CATALOG routine that prints the volume number for a disk. See PRTDEC ($AE42).

Actual Location of Reference: $AE09 (STA $45)
Part of the Routine: FILESIZ ($AE01)
Purpose/Comments: Prints file size of each file during a CATALOG (see PRTDEC - $AE42).

Actual Location of Reference: $AE53 (LDA $45)
Part of the Routine: PRTDEC ($AE42)
Purpose/Comments: Prints decimal number based on integer bytes in $44,45. Used by DOS a lot, and occasionally by programmers in need of a decimal printing routine!

Actual Location of Reference: $BED2 (STA $45)
Part of the Routine: DSKFRMT ($BEAF)
Purpose/Comments: Part of INIT routine.

Actual Location of Reference: $BF15 (LDY $45)
Part of the Routine: TRKFRMT ($BF15)
Purpose/Comments: Formats a given track during an INIT.

Actual Location of Reference: $BF38 (LDY $45)
Part of the Routine: INITMAP ($BF32)
Purpose/Comments: Marks a track as formatted. Still part of INIT.

Actual Location of Reference: $BF54,BF56,BF5A
(CMP, LDA, STA $45)
Part of the Routine: INITMAP ($BF32)
Purpose/Comments: Still part of INIT.

Actual Location of Reference: $BF9C,BFA2,BFA4
(LDA, LDA, DEC $45)
Part of the Routine: MARKMAP ($BF8B)
Purpose/Comments: Part of track format part of INIT.

As you can see from all this, the inescapable conclusion is that you just shouldn't use DOS while using interrupts - even with a clock card! The only real solution is to modify the F8 Monitor ROM so that location $45 isn't destroyed by an interrupt. This solves all the problems at once, and is described in Appendix B.

APPENDIX B: MODIFIED F8 ROM OPTION

There are two ways of looking at (and solving) the problem of the conflict between DOS and the Monitor's use of location $45 and the Monitor's IRQ handler.

Modifying DOS 3.3 and the Monitor ROM

The first is to say that DOS and the Monitor should not use location $45 for their general routines. You could alter the Monitor and DOS to solve this. The main problems here are that in addition to altering the Monitor ROM (also called the F8 ROM) you must also modify DOS 3.3 to eliminate all uses of location $45 and then boot on this custom version of DOS. If you are running software off another diskette that you must boot on, you will not be able to run your interrupt software at the same time.

In addition, ProDOS checks the Monitor on boot, and will not boot on your modified Monitor ROM.

Modifying just the Monitor ROM IRQ Routine

The other option is to again alter the Monitor, but in this case, only change the IRQ handler so as to not destroy location $45. This still suffers from the problem of the ProDOS boot check, but does eliminate the need for a custom DOS 3.3, and avoids conflict with programs that others may write that use location $45 as a temporary register.

Let's take a second look at how the Monitor (Auto-Boot version) usually handles an interrupt:

```
FA40-  85 45       STA $45        ; save Acc.
FA42-  68          PLA            ; retrieve Status Reg.
FA43-  48          PHA            ; copy Status to stack
FA44-  0A          ASL
FA45-  0A          ASL
FA46-  0A          ASL            ; shift int flg to N
FA47-  30 03       BMI $FA4C      ; branch if BRK
FA49-  6C FE 03    JMP ($3FE)     ; JMP via vector
FA4C-  28          PLP            ; retrieve Status Reg.
FA4D-  20 4C FF    JSR $FF4C      ; save regs (not Acc.)
```

The first thing that happens when an interrupt occurs is that the Accumulator is stored in location $45. This is the source of the location $45 conflict under interrupts. After testing the BRK flag, the routine then either jumps to an address pointed to by $3FE,3FF (if an interrupt) or falls through to $FA4C to process the BRK instruction.

Suppose for a moment that we were to re-write this part of the Monitor to read as follows:

```
FA40-  6C FE 03     JMP ($3FE)      ; jump to interrupt
                                      routine.
FA43-  85 45        STA $45         ; save acc.
FA45-  68           PLA             ; retrieve Status Reg.
FA46-  48           PHA             ; restore stack
FA47-  0A           ASL
FA48-  0A           ASL
FA49-  0A           ASL             ; shift BRK flag into
                                      sign bit
FA4A-  10 13        BPL $FA5F       ; branch to a jump to
                                      the Monitor
FA4C-  28           PLP             ; retrieve Status Reg.
                                     (beg. of BRK routine)
```

In this version, we jump immediately to our own interrupt routine pointed to by $3FE,3FF, and can then save the Accumulator in a location of our own chosing, and that is presumably more compatible with the rest of the Apple (such as the internal storage locations used by routines in this package).

Under normal condition, DOS stores the address $FF65 in locations $3FE,3FF during the boot process. This points to the jump-to-monitor routine in the Apple Monitor. ($FF65 is roughly the same as a CALL -151 from BASIC).

With the modified ROM, if an interrupt is received, control immediately goes to $FA40 (via the vector at $FFFE,FFFF), which then jumps via the vector at $3FE,3FF. With locations $3FE,3FF set to their boot-up address ($FF65), control is then passed to $FF65 with the result being a jump to the Monitor (without the

```
"beep!" 1234- A=00 X=23 Y=FF P=DE S=67
```

type of message).

If you do not modify DOS, then the instructions from $FA43 through $FA4C will never be executed, and BRKs will be treated just like IRQs, i.e. an immediate jump to the Monitor with no particular message. (Remember that both IRQs and BRKs are vectored to $FA40 by the bytes at $FFFE,FFFF).

By modifying DOS on the disk, you can make DOS set $3FE,3FF to point to $FA43 instead of $FF65 at boot-up time.

To modify DOS on a disk, you can use a disk editing utility to change bytes $7F and $80, track 0, sector $0D from $65,FF to $43,FA. You can also change locations $9E7F,9E80 in memory from $65,FF to $43,FA and then initialize a blank disk.

Locations $3FE,3FF can also be set to point to $FA43 with a set of POKEs (or STA's) at any time after boot up as well. In that case, the instructions from $FA43 through $FA4C will handle BRK instructions and interrupts just as the unmodified Monitor ROM would, i.e. Monitor messages on a BRK, and a message-less jump to Monitor on an IRQ.

IMPLEMENTING THE F8 ROM MODIFICATIONS

The usual way of modifying part of the ROM in the Apple is to program an EPROM (Erasable Programmable Read Only Memory) with the new version of the software that usually occupies that portion of memory. This is done using "EPROM Burner" and requires some special equipment.

If you are not familiar with, or do not have access to this technique, you may want to consider purchasing a pre-programmed substitute F8 ROM chip, which is described in the following section, "DOUBLETIME PRINTER OFFER".

If you do want to program your own EPROM, just substitute the code listed above in the range of $FA40 through $FA4C for the equivalent bytes in the normal F8 Monitor ROM code.

A couple of notes here. First, the ROMs that the Apple II/II+ use are almost, but not quite, compatible with the popular 2716 EPROM (Eraseable Programmable Read Only Memory). The difference is in the polarity of the signals on pins 18 and 21. To our knowledge there are five ways to get around this incompatibility.

1) Bend up pins 18 and 21 and solder them to wires running to pins 24 and 12, respectively. This is not recommended because it overrides the Inhibit signal that turns off the ROM addresses.

2) Wire an adapter that passes through all signals except those on pins 18 and 21, and cross wire those so that the signal that is on pin 18 of the APPLE socket appears on pin 21 of the EPROM and vice versa. This system works most of the time on most Apples, but may not work with all RAM cards. The reason for this problem is that Apple uses pin 18 for INHIBIT to keep the ROMs from being enabled when the ROM address space is being used by a ROM or RAM card. Pin 21 on the 2716 EPROM is not a timed signal. The 2716 expects to see a solid 5 volts. Inhibit is a timed signal, and thus can be "missed" by the 2716.

3) The PROMETTE by Computer Microworks, Inc, P.O. Box 33651, Dayton, Ohio, 45433, (305) 777-0268 is a socket for 2716s that has an inverter chip on board that "fixes" the signals to make 2716s Apple compatible.

4) Have a chip manufacturer make a special ROM chip to your specifications. This has the drawbacks of costing over $1500 and taking three months.

5) Take advantage of the DOUBLETIME PRINTER OFFER mentioned in this manual. This includes an Apple compatible ROM already programmed to fix the IRQ location $45 problem. The only disadvantage here is that ProDOS checks to see if the F8 ROM is an "official" Apple ROM when it boots, and if the ROM doesn't check out (i.e. not identical to the usual F8 AutoBoot ROM), ProDOS won't boot. Not very nice on their part.

There is a patch to this however. If you have a ProDOS track/sector edit utility that lets you change data on the diskette, change bytes $60-61 on track 1, sector 9 from $A9 00 to $A5 0C. This disables the ROM check routine.

It may also be necessary to change track 1, sector $C, bytes $B4-B6 from AE B3 FB to A2 EA EA. This tells ProDOS that your computer is a II+. Making byte $B5 an $A0 tells it you have a //e.

PROGRAMMING INTERRUPT ROUTINES WITH THE MODIFIED ROM

Using the modified ROM gives the interrupt programmer more freedom in preserving and restoring registers. The interrupt routine, whose address is stored at $3FE,3FF, must save all registers it uses and must determine whether it has been called by an IRQ or a BRK. (See the code at $FA43 for the technique to do this.)

Just remember to save and restore the Accumulator froma location not used by the foreground task. Don't restore the Accumulator from location $45 if you're using a modified ROM. Do restore it from $45 if your ROM is the Apple standard Monitor ROM.

THE UNIVERSAL INTERRUPT RETURN

The listing on the next page is a listing for an alternate exit for interrupt routines so that they will work on either standard or modified (as described above) F8 Monitor ROMs. In altering any of the source listings in this kit for example, you would just replace everything from the label RETURN on down in each source listing with the RETURN routine listed here.

The idea behind this return technique is to test location $FA40 in the Monitor to see if the byte there corresponds to a standard or the modified F8 ROM. If location $FA40 holds the value $85, it is a standard ROM, and the routine falls through the branch to load the Accumulator with the contents of location $45 before returning with the usual RTI.

If location $FA40 isn't equal to $85, the Monitor is probably a modified ROM, and so the routine branches to RETURN1 where the Accumlator is restored using the internal storage location SAVREGS.

When altering your routines to use the universal interrupt return, you do have to remember to store the contents of the accumulator in SAVREGS or its equivalent at the entry to your routine. This already done for you in all the listings in this kit, so it is only necessary to add the new RETURN routine at the end of them.

```
                        1    ********************************
                        2    * UNIVERSAL INTERRUPT RETURN   *
                        3    ********************************
                        4
                        5             ORG  $300
                        6             OBJ  $300
                        7
                        8    MONIRQ   EQU  $FA40      ; ADDR. OF MONITOR
                                                        IRQ ROUTINE
                        9
0300: 00                10    SAVREGS  HEX  00        ; STORE ACC.
                        11
0301: AD 40 FA          12    RETURN   LDA  MONIRQ    ; CHECK MONITOR ROM
0304: C9 85             13             CMP  #$85      ; IS THIS DBLTM ROM?
0306: D0 03             14             BNE  RETURN1   ; YES!
0308: A5 45             15             LDA  $45       ; RESTORE A
030A: 40                16             RTI
                        17
030B: AD 00 03          18    RETURN1  LDA  SAVREGS   ; RESTORE ACC FROM
                                                        SAVREG
030E: 40                19    RETURN2  RTI
```

DOUBLETIME PRINTER OFFER

(Apple II/II+ only)

If you have an Apple II or II+ (not //e or //c) and don't want to make your own EPROM, or want to avoid the possible problems related to EPROMs and RAM cards, we have a special offer to purchasers of The Interrupt Experimenter's Kit.

Doubletime Printer is an interrupt driven print spooling package that will print files from disk under DOS 3.3 while you are running a program in the foreground. You can even list Applesoft programs directly from disk.

The product also includes a special modified F8 ROM that has the interrupt routine patch described in this section. Since it's a ROM you don't have to worry about problems with RAM cards, and we also fixed the lower case input problem while we were changing things.

This means that when you copy over lower case text on the screen, it won't be converted to upper case. Also, when the cursor is on an upper case letter on the screen, it will be displayed as an inverse capital letter, as opposed to the strange %@#&, etc. characters that the usual Apple II/II+ machines display.

Doubletime Printer comes with the modified F8 ROM, a 70 page instruction manual, and a diskette with the Doubletime Printer software.

To receive your Doubletime Printer package, send a check or money order for $25.00 ($28.00 in Calif.) to:

Doubletime Printer Special
Roger Wagner Publishing, Inc.
P.O. Box 582-I
Santee, CA 92071
(619) 562-3670

You can also use Mastercharge and Visa, just be sure to include your expiration date and card number with your order.

APPENDIX C: EXTRA DEMONSTRATION PROGRAMS

There are a number of other demonstration programs provided on The Interrupt Experimenter's Kit diskette.

They are:

1) LO-RES COLOR SHIFTER: RUN the program LORES PROGRAM to see how an interrupt driven routine can be combined with Applesoft.

The screen image should change colors as the interrupt routine changes every color pixel every 10th of a second or so. Notice that you can type anything you want in the window at the bottom of the screen, including any normal Applesoft command like PRINT "HELLO", LIST, etc. while the interrupts operate on the screen "in the background". Press RESET to stop the interrupts and return the screen to the normal TEXT mode.

You may also want to re-assemble the routine with a faster (less than 20) RATE value to speed up the screen action, and observe the corresponding slowing down of the foreground operations. You can also try changing the RATE value by adding a POKE 779, K to the LORES PROGRAM (at line 45 for example), where K is a number between 1 and 50 that determines the rate of the screen cycles.

2) PONG: BRUN PONG from the immediate mode of Applesoft. The bottom portion of the screen will be protected from normal scrolling, and a "ball" will bounce around inside the box that is drawn at the bottom of the screen. Again, notice that you can LIST programs, etc. Pressing RESET will stop the interrupt program.

3) BOUNCER: This is similar to the PONG program, except in this case a border is not drawn on the screen, and the "ball" is free to bounce anywhere on the screen while you LIST your program, CATALOG the disk, etc. When the screen scrolls, the ball leaves a trail because the program cannot erase the ball when it draws a new one. This is because the interrupt program has no way of knowing that the screen has scrolled. Remember that interrupt programs are generally independent of any foreground programs or activities that are going on. Pressing RESET will stop the BOUNCER program.

```
                        1    ********************************
                        2    *INTERRUPT DRIVEN COLOR DISPLAY*
                        3    *            1/21/85           *
                        4    ********************************
                        5
                        6    * CHANGES PLOTTED COLORS ONLY ON A LORES
                               SCREEN
                        7
                        8    IRQ       EQU  $3FE
                        9    PTR       EQU  $6          ; Z-PG LOCS $06,07
                        10   MONIRQ    EQU  $FA40       ; LOC OF NORMAL MON
                                                          IRQ ROUTINE
                        11
                        12             OBJ  $300
                        13             ORG  $300
                        14
                        15   * SET UP INTERRUPT VECTOR
                        16
0300: A9 1E             17   INIT      LDA  #<START     ; LO BYTE OF IRQ
                                                          ROUTINE ADDRESS
0302: 8D FE 03          18             STA  IRQ         ; PUT IN LO BYTE OF
                                                          VECTOR
0305: A9 03             19             LDA  #>START     ; HI BYTE OF IRQ
                                                          ROUTINE ADDRESS
0307: 8D FF 03          20             STA  IRQ+1       ; PUT IN HI BYTE OF
                                                          VECTOR
030A: A9 1E             21             LDA  #20         ; # OF INTS TO WAIT
                                                          PER CYCLE
030C: 8D 17 03          22             STA  RATE        ; INITIALIZE RATE =
                                                          20
030F: 8D 18 03          23             STA  TIME        ; SET TIME TO RATE
                        24
                        25   * ENABLE INTS. AND TURN ON CARD
                        26
0312: 58                27             CLI              ; ENABLE INTERRUPTS
0313: AD C0 C0          28             LDA  $C0C0       ; $C0(8+N)0 = TURN
                                                          ON CARD
0316: 60                29             RTS              ; STARTUP DONE!
                        30
                        31   * VARIABLES
                        32
0317: 00                33   RATE      HEX  00          ; # OF INTS PER
                                                          SCREEN CYCLE
0318: 00                34   TIME      HEX  00          ; CURRENT INT #
                                                          COUNTER
0319: 00                35   VALUE     HEX  00          ; TEMP VALUE FOR A
                                                          NIBBLE
031A: 00                36   YP        HEX  00          ; Y POSN ON SCREEN
                                                          (0-39)
                        37
```

```
031B: 00          38  SAVREGS  HEX  00          ; LOCATION TO SAVE
                                                   ACC.
031C: 00          39           HEX  00          ; LOCATION TO SAVE
                                                   X-REGISTER
031D: 00          40           HEX  00          ; LOCATION TO SAVE
                                                   Y-REGISTER
                  41
031E: 8D 1B 03    42  START    STA  SAVREGS     ; SAVE ACCUMULATOR
                                                   VALUE
0321: 8E 1C 03    43           STX  SAVREGS+1   ; SAVE X REG
0324: 8C 1D 03    44           STY  SAVREGS+2   ; SAVE Y REG
                  45
                  46  * DEC TIME. READY YET?
                  47  * IF SO, RESET TIME COUNT & CONT
                  48  * ELSE, RETURN
                  49
0327: CE 18 03    50           DEC  TIME        ; TIME = TIME - 1
032A: D0 49       51           BNE  RETURN      ; BACK TO FOREGROUND
                                                   IF NOT DONE
032C: AD 17 03    52           LDA  RATE        ; GET 'RATE' VALUE
032F: 8D 18 03    53           STA  TIME        ; RESTORE 'TIME'
                                                   COUNTER
0332: A9 13       54           LDA  #19         ; 19 = LAST LINE ON
                                                   GR SCREEN
0334: 8D 1A 03    55           STA  YP          ; SET YP TO LINE 19
                  56
                  57  * MAIN LOOP :
                  58  * FIRST GET BASE ADDRESS FOR
                  59  * YPOS, THEN LOOP ACROSS
                  60  * ROW. WHEN DONE, GOTO NEXT
                  61  * YPOS POSITION.
                  62
0337: AC 1A 03    63  GL       LDY  YP          ; GET LINE # TO WORK
                                                   ON
033A: B9 89 03    64           LDA  ADDRLO,Y    ; GET LO BYTE OF
                                                   BASE ADDRESS
033D: 85 06       65           STA  PTR         ; PUT IN PTR
033F: B9 A1 03    66           LDA  ADDRHI,Y    ; GET HI BYTE OF
                                                   BASE ADDRESS
0342: 85 07       67           STA  PTR+1       ; PUT IN PTR+1
0344: A0 27       68           LDY  #39         ; SET Y = END OF
                                                   SCREEN LINE
                  69
                  70  * DO EACH NIBBLE AT A TIME
                  71  * DEC THE NIBBLE. IF 0, RESET
                  72  * TO $F (15).
                  73
0346: B1 06       74  GTOP     LDA  (PTR),Y     ; GET SCRN VALUES
                                                   FOR TOP & BOTTOM
0348: F0 23       75           BEQ  NXTBLK      ; SKIP IF BOTH BLACK
034A: 29 0F       76           AND  #$0F        ; LOOK AT TOP BLOCK
```

```
034C: 8D 19 03   77              STA  VALUE      ; SAVE TOP VALUE
034F: F0 0A      78              BEQ  GBOTM      ; SKIP IF BLACK
0351: CE 19 03   79              DEC  VALUE      ; COLOR = COLOR - 1
0354: D0 05      80              BNE  GBOTM      ; IF NOT BLACK
0356: A9 0F      81              LDA  #$0F       ; SET BACK TO 15 =
                                                   WHITE
0358: 8D 19 03   82              STA  VALUE      ; SAVE VALUE
                 83
035B: B1 06      84   GBOTM     LDA  (PTR),Y    ; GET BYTE AGAIN
035D: 29 F0      85              AND  #$F0       ; LOOK AT BOTTOM
                                                   BLOCK
035F: F0 07      86              BEQ  MIX        ; FINISH IF BLACK
0361: 38         87              SEC
0362: E9 10      88              SBC  #$10       ; COLOR = COLOR - 1
                                                   (NO - 10)
                 89                              ; REMEMBER THIS IS
                                                   BECAUSE
                 90                              ; WE'RE DOING THE
                                                   UPPER NIBBLE
0364: D0 02      91              BNE  MIX        ; IF NOT BLACK
0366: A9 F0      92              LDA  #$F0       ; RESET TO BLACK
                 93
0368: 0D 19 03   94   MIX       ORA  VALUE      ; ENTER WITH ACC =
                                                   BOTTOM NIBBLE
036B: 91 06      95              STA  (PTR),Y    ; PUT NEW COLOR ON
                                                   SCREEN
                 96
                 97   * END OF LOOP FOR X LOOP
                 98   * AND Y LOOP.
                 99
036D: 88         100  NXTBLK    DEY             ; Y = Y - 1 FOR NEXT
                                                   HORIZ.  POSN.
036E: 10 D6      101             BPL  GTOP       ; DO NEXT BOX IF NOT
                                                   DONE
0370: CE 1A 03   102             DEC  YP         ; YP = YP - 1 FOR
                                                   NEXT SCREEN LINE
0373: 10 C2      103             BPL  GL         ; PROCESS NEXT LINE
                                                   IF NOT DONE
                 104
                 105  * WHEN DONE RESTORE REGISTERS
                 106  * AND RETURN
                 107
0375: AE 1C 03   108  RETURN    LDX  SAVREGS+1  ; RESTORE X REGISTER
0378: AC 1D 03   109             LDY  SAVREGS+2  ; RESTORE Y REGISTER
037B: AD 40 FA   110             LDA  MONIRQ     ; CHECK MONITOR ROM
                                                   ID BYTE
037E: C9 85      111             CMP  #$85       ; DBLTM ROM?
0380: D0 03      112             BNE  RETURN1    ; YES!
0382: A5 45      113             LDA  $45        ; RESTORE ACC. FROM
                                                   LOC. $45
```

```
0384: 40          114            RTI              ; RETURN FROM THE
                                                    INTERRUPT
                  115
0385: AD 1B 03    116  RETURN1   LDA  SAVREGS     ; RESTORE ACC FROM
                                                    SAVREGS
0388: 40          117  RETURN2   RTI              ; RETURN FROM THE
                                                    INTERRUPT
                  118
                  119  ** ADDRESSES OF 1ST BYTE OF EACH SCREEN LINE
                  120
0389: 00 80 00    121  ADDRLO    HEX  0080008000
038C: 80 00
038E: 80 00 80    122            HEX  80008028A8
0391: 28 A8
0393: 28 A8 28    123            HEX  28A828A828
0396: A8 28
0398: A8 50 D0    124            HEX  A850D050D0
039B: 50 D0
039D: 50 D0 50    125            HEX  50D050D0
03A0: D0
03A1: 04 04 05    126  ADDRHI    HEX  0404050506
03A4: 05 06
03A6: 06 07 07    127            HEX  0607070404
03A9: 04 04
03AB: 05 05 06    128            HEX  0505060607
03AE: 06 07
03B0: 07 04 04    129            HEX  0704040505
03B3: 05 05
03B5: 06 06 07    130            HEX  06060707
03B8: 07
```

```
                        1    ********************************
                        2    *    INTERRUPT DRIVEN PONG     *
                        3    *          1/21/85             *
                        4    ********************************
                        5
                        6    IRQ       EQU  $3FE
                        7    PTR       EQU  $06          ; $06,07
                        8    BOTTOM    EQU  $23
                        9    MONIRQ    EQU  $FA40
                        10   SPKR      EQU  $C030
                        11   HOME      EQU  $FC58
                        12
                        13             ORG  $8000
                        14
                        15   * SET INT. VECTORS
                        16   * CLEAR SCREEN AND SET TEXT WIND.
                        17   * SET X & Y DELTAS, X & Y POS,
                        18   * AND DRAW INVERSE BORDER
                        19
8000: A9 91             20   INIT      LDA  #<START      ; LO BYTE OF IRQ
                                                           ROUTINE ADDR.
8002: 8D FE 03          21             STA  IRQ          ; PUT IN LO BYTE OF
                                                           VECTOR
8005: A9 80             22             LDA  #>START      ; HI BYTE OF IRQ
                                                           ROUTINE
8007: 8D FF 03          23             STA  IRQ+1        ; PUT IN HI BYTE OF
                                                           VECTOR
800A: 20 58 FC          24             JSR  HOME         ; CLEAR SCREEN &
                                                           HOME CURSOR
800D: A9 0F             25             LDA  #15          ; 15 = TOP OF "GAME"
                                                           AREA
800F: 85 23             26             STA  BOTTOM       ; SET TEXT SCROLL
                                                           WINDOW
8011: A9 FF             27             LDA  #$FF         ; #$FF = -1 IN 2'S
                                                           COMPLEMENT MATH
8013: 8D 89 80          28             STA  XD           ; SET X VELOCITY =
                                                           -1
8016: 8D 8A 80          29             STA  YD           ; SET Y VELOCITY =
                                                           -1
8019: A9 14             30             LDA  #20          ; # OF INTS PER
                                                           CYCLE
801B: 8D 85 80          31             STA  RATE         ; SET RATE VALUE
801E: 8D 86 80          32             STA  TIME         ; SET TIME = RATE
8021: A9 14             33             LDA  #20          ; STARTING X,Y POSN
                                                           FOR BALL
8023: 8D 87 80          34             STA  XPOS         ; X POSN = 20
8026: 8D 88 80          35             STA  YPOS         ; Y POSN = 20
8029: A9 27             36             LDA  #39          ; 39 = RT SIDE OF
                                                           SCREEN
802B: 8D 8C 80          37             STA  XP           ; SET UP TO DRAW
                                                           WALLS
```

```
802E: A9 0F      38              LDA  #15        ; 15 = TOP OF "GAME"
                                                   AREA
8030: 8D 8D 80   39              STA  YP         ; SET UP TO DRAW
                                                   WALLS
8033: A9 20      40              LDA  #$20       ; VALUE FOR AN
                                                   INVERSE BOX
8035: 8D 8B 80   41              STA  VALUE      ; VALUE FOR PLOT
                                                   ROUTINE
                 42
8038: 20 5D 81   43  DRAWTOP     JSR  PLOT       ; DRAW TOP WALL
803B: CE 8C 80   44              DEC  XP         ; XP = XP - 1 UNTIL
                                                   DONE
803E: 10 F8      45              BPL  DRAWTOP    ; LOOP FOR A LINE
8040: A9 27      46              LDA  #39        ; 39 = RT EDGE OF
                                                   SCREEN
8042: 8D 8C 80   47              STA  XP         ; XP = 39
8045: A9 17      48              LDA  #23        ; 23 = BOTTOM OF
                                                   SCREEN
8047: 8D 8D 80   49              STA  YP         ; YP = 23
                 50
804A: 20 5D 81   51  DRAWBTM     JSR  PLOT       ; DRAW BOTTOM WALL
804D: CE 8C 80   52              DEC  XP         ; XP = XP - 1 UNTIL
                                                   DONE
8050: 10 F8      53              BPL  DRAWBTM    ; LOOP FOR A LINE
8052: A9 00      54              LDA  #0         ; 0 FOR LEFT EDGE OF
                                                   SCREEN
8054: 8D 8C 80   55              STA  XP         ; XP = 0
8057: A9 10      56              LDA  #16        ; 16 = LINE FOR TOP
                                                   OF WALL
8059: 8D 8D 80   57              STA  YP         ; YP = 16
                 58
805C: 20 5D 81   59  DRAWLFT     JSR  PLOT       ; DRAW THE LEFT WALL
805F: EE 8D 80   60              INC  YP         ; YP = YP + 1
8062: AD 8D 80   61              LDA  YP         ; GET VALUE TO CHECK
8065: C9 19      62              CMP  #25        ; 25 = BOTTOM OF
                                                   SCREEN + 1
8067: D0 F3      63              BNE  DRAWLFT    ; DRAW UNTIL DONE
8069: A9 27      64              LDA  #39        ; 39 = RT SIDE OF
                                                   SCREEN
806B: 8D 8C 80   65              STA  XP         ; XP = 39
806E: A9 10      66              LDA  #16        ; 16 = LINE FOR TOP
                                                   OF WALL
8070: 8D 8D 80   67              STA  YP         ; YP = 16
8073: 20 5D 81   68  DRAWRT      JSR  PLOT       ; DRAW THE RT SIDE
                                                   WALL
8076: EE 8D 80   69              INC  YP         ; YP = YP + 1
8079: AD 8D 80   70              LDA  YP         ; GET VALUE TO CHECK
807C: C9 19      71              CMP  #25        ; 25 = BOTTOM OF
                                                   SCREEN + 1
807E: D0 F3      72              BNE  DRAWRT     ; DRAW UNTIL DONE
                 73
```

```
                        74   * ENABLE INTS. AND ACTIVATE CARD
                        75
8080: 58                76            CLI              ; ENABLE INTERRUPT
                                                         FLAG
8081: AD C0 C0          77            LDA  $C0C0       ; $C0(8+N)0 = TURN
                                                         ON CARD
8084: 60                78            RTS              ; STARTUP IS DONE!
                        79
                        80   * VARIABLES
                        81
8085: 00                82   RATE     HEX  00          ; # OF INTS PER
                                                         SCREEN CYCLE
8086: 00                83   TIME     HEX  00          ; CURRENT INT #
                                                         COUNTER
8087: 00                84   XPOS     HEX  00          ; CURRENT X POSN OF
                                                         BALL
8088: 00                85   YPOS     HEX  00          ; CURRENT Y POSN OF
                                                         BALL
8089: 00                86   XD       HEX  00          ; VALUE FOR X
                                                         VELOCITY
808A: 00                87   YD       HEX  00          ; VALUE FOR Y
                                                         VELOCITY
808B: 00                88   VALUE    HEX  00          ; VALUE FOR PLOT
                                                         ROUTINE TO DRAW
808C: 00                89   XP       HEX  00          ; X COORDINATE FOR
                                                         PLOT ROUTINE
808D: 00                90   YP       HEX  00          ; Y COORDINATE FOR
                                                         PLOT ROUTINE
808E: 00                91   SAVREGS  HEX  00          ; LOCATION TO SAVE
                                                         ACC
808F: 00                92            HEX  00          ; LOCATION TO SAVE
                                                         X-REG
8090: 00                93            HEX  00          ; LOCATION TO SAVE
                                                         Y-REG
                        94
                        95   * DISABLE INTS. & SAVE REGS
                        96
8091: 8D 8E 80          97   START    STA  SAVREGS     ; SAVE ACCUMULATOR
8094: 8E 8F 80          98            STX  SAVREGS+1   ; SAVE X REGISTER
8097: 8C 90 80          99            STY  SAVREGS+2   ; SAVE Y REGISTER
                        100
                        101  * TIMER TIMED OUT?
                        102  * IF NOT, RETURN. IF SO, RESET
                        103  * TIMER & CONTINUE
                        104
809A: CE 86 80          105           DEC  TIME        ; TIME = TIME - 1
809D: D0 6C             106           BNE  RETURN      ; RETURN TO
                                                         FOREGROUND IF NOT
                                                         DONE
809F: AD 85 80          107           LDA  RATE        ; GET 'RATE' VALUE
```

```
80A2: 8D 86 80  108           STA  TIME       ; RESTORE TIME
                                                COUNTER
                109
                110  * ERASE BALL
                111
80A5: A9 A0     112           LDA  #" "       ; SPACE CHARACTER
80A7: 8D 8B 80  113           STA  VALUE      ; PUT IN VALUE FOR
                                                PLOT
80AA: AD 87 80  114           LDA  XPOS       ; GET CURRENT X POSN
                                                OF BALL
80AD: 8D 8C 80  115           STA  XP         ; XP = XPOS
80B0: AD 88 80  116           LDA  YPOS       ; GET CURRENT Y POSN
                                                OF BALL
80B3: 8D 8D 80  117           STA  YP         ; YP = YPOS
80B6: 20 5D 81  118           JSR  PLOT       ; ERASE THE BALL
                119
                120  * ADD DELTAS TO CURRENT POSITION
                121
80B9: AD 87 80  122  NEWX     LDA  XPOS       ; GET CURRENT X POSN
                                                OF BALL
80BC: 18        123           CLC             ; CLEAR CARRY FOR AN
                                                ADD
80BD: 6D 89 80  124           ADC  XD         ; ADD X VELOCITY
                125
                126  * CHECK FOR HITTING BOUNDARIES
                127  * IF THEY HIT, REVERSE DELTA
                128  * DIRECTION
                129
80C0: F0 0A     130           BEQ  RVRSX      ; IF X = 0 THEN
                                                REVERSE
80C2: C9 27     131           CMP  #39        ; 39 = RIGHT WALL
80C4: F0 06     132           BEQ  RVRSX      ; REVERSE IF THERE
80C6: 8D 87 80  133           STA  XPOS       ; PUT VALUE BACK IF
                                                OK
80C9: 4C D7 80  134           JMP  NEWY       ; CALCULATE NEW Y
                                                POSN
80CC: AD 89 80  135  RVRSX    LDA  XD         ; GET X VELOCITY
80CF: 49 FE     136           EOR  #$FE       ; XD = XD * -1 TO
                                                REVERSE MOTION
80D1: 8D 89 80  137           STA  XD         ; PUT BACK
80D4: 20 4F 81  138           JSR  SOUND      ; MAKE THE 'BOUNCE'
                                                NOISE
                139
80D7: AD 88 80  140  NEWY     LDA  YPOS       ; GET CURRENT Y POSN
                                                OF BALL
80DA: 18        141           CLC             ; CLEAR CARRY FOR AN
                                                ADD
80DB: 6D 8A 80  142           ADC  YD         ; ADD Y VELOCITY
                                                VALUE
80DE: C9 0F     143           CMP  #15        ; 15 = TOP WALL
```

```
80E0: F0 0A      144              BEQ  RVRSY      ; REVERSE VELOCITY
                                                   IF THERE
80E2: C9 17      145              CMP  #23       ; 23 = BOTTOM WALL
80E4: F0 06      146              BEQ  RVRSY     ; REVERSE VELOCITY
                                                   IF THERE
80E6: 8D 88 80   147              STA  YPOS      ; PUT BACK IF VALUE
                                                   OK
80E9: 4C F7 80   148              JMP  DRAW      ; DRAW THE NEW BALL
                                                   IMAGE
80EC: AD 8A 80   149  RVRSY      LDA  YD        ; GET CURRENT Y
                                                   VELOCITY
80EF: 49 FE      150              EOR  #$FE      ; YD = YD * -1 TO
                                                   REVERSE MOTION
80F1: 8D 8A 80   151              STA  YD        ; PUT BACK NEW
                                                   VELOCITY
80F4: 20 4F 81   152              JSR  SOUND     ; MAKE THE 'BOUNCE'
                                                   NOISE
                 153
                 154  * FINALLY, DRAW THE BALL
                 155
80F7: AD 87 80   156  DRAW       LDA  XPOS      ; GET CURRENT X POSN
                                                   OF BALL
80FA: 8D 8C 80   157              STA  XP        ; PUT IN XP FOR PLOT
                                                   ROUTINE
80FD: AD 88 80   158              LDA  YPOS      ; GET CURRENT Y POSN
                                                   OF BALL
8100: 8D 8D 80   159              STA  YP        ; PUT IN YP FOR PLOT
                                                   ROUTINE
8103: A9 CF      160              LDA  #"O"      ; CHARACTER FOR THE
                                                   BALL
8105: 8D 8B 80   161              STA  VALUE     ; PUT IN VALUE TO
                                                   PLOT
8108: 20 5D 81   162              JSR  PLOT      ; DRAW THE NEW BALL
                                                   IMAGE
810B: AE 8F 80   163  RETURN     LDX  SAVREGS+1 ; RESTORE X
810E: AC 90 80   164              LDY  SAVREGS+2 ; RESTORE Y
8111: AD 40 FA   165              LDA  MONIRQ    ; CHECK MONITOR ROM
8114: C9 85      166              CMP  #$85      ; DBLTM ROM?
8116: D0 03      167              BNE  RETURN1   ; YES
8118: A5 45      168              LDA  $45       ; RESTORE ACC FROM
                                                   LOC. $45
811A: 40         169              RTI            ; RETURN FROM THE
                                                   INTERRUPT
                 170
811B: AD 8E 80   171  RETURN1    LDA  SAVREGS   ; RESTORE ACC FROM
                                                   SAVREGS
811E: 40         172              RTI            ; RETURN FROM THE
                                                   INTERRUPT
                 173
                 174  ** ADDRESSES OF 1ST BYTE OF EACH SCREEN LINE
                 175
```

```
811F: 00 80 00  176 ADDRLO   HEX  0080008000
8122: 80 00
8124: 80 00 80  177           HEX  80008028A8
8127: 28 A8
8129: 28 A8 28  178           HEX  28A828A828
812C: A8 28
812E: A8 50 D0  179           HEX  A850D050D0
8131: 50 D0
8133: 50 D0 50  180           HEX  50D050D0
8136: D0
8137: 04 04 05  181 ADDRHI   HEX  0404050506
813A: 05 06
813C: 06 07 07  182           HEX  0607070404
813F: 04 04
8141: 05 05 06  183           HEX  0505060607
8144: 06 07
8146: 07 04 04  184           HEX  0704040505
8149: 05 05
814B: 06 06 07  185           HEX  06060707
814E: 07
                186
                187 * BOUNCE SOUND
                188
814F: A2 0A     189 SOUND    LDX  #10        ; VALUE FOR DURATION
                                               TONE
8151: A0 32     190 DLOOP    LDY  #50        ; VALUE FOR PITCH OF
                                               TONE
8153: AD 30 C0  191          LDA  SPKR       ; CLICK THE SPEAKER
                                               ONCE
8156: 88        192 PLOOP    DEY             ; PITCH COUNTER
8157: D0 FD     193          BNE  PLOOP      ; LOOP FOR PITCH
8159: CA        194          DEX             ; DURATION COUNTER
815A: D0 F5     195          BNE  DLOOP      ; LOOP FOR DURATION
815C: 60        196          RTS             ; DONE WITH THE TONE
                197
                198 * TEXT PLOT, SAVING BCKRND
                199
815D: AC 8D 80  200 PLOT     LDY  YP         ; GET LINE # TO PLOT
                                               ON
8160: B9 1F 81  201          LDA  ADDRLO,Y   ; GET LO BYTE OF
                                               ADDRESS
8163: 85 06     202          STA  PTR        ; PUT IN PTR
8165: B9 37 81  203          LDA  ADDRHI,Y   ; GET HI BYTE OF
                                               ADDRESS
8168: 85 07     204          STA  PTR+1      ; PUT IN PTR+1
816A: AC 8C 80  205          LDY  XP         ; GET HORIZ. POSN ON
                                               LINE
816D: AD 8B 80  206          LDA  VALUE      ; GET VALUE TO PLOT
8170: 91 06     207          STA  (PTR),Y    ; PUT IT ON THE
                                               SCREEN
8172: 60        208          RTS             ; DONE WITH THE PLOT
```

```
                        1    ********************************
                        2    *INTERRUPT DRIVER BOUNCING BALL*
                        3    *              1/21/85          *
                        4    ********************************
                        5
                        6    IRQ       EQU  $3FE
                        7    PTR       EQU  $06,07
                        8    MONIRQ    EQU  $FA40
                        9    HOME      EQU  $FC58       ; MONITOR HOME
                                                          ROUTINE
                       10
                       11    SPKR      EQU  $C030
                       12
                       13              ORG  $8000
                       14
                       15    * SET INT. VECTOR
                       16
8000: A9 52            17    INIT      LDA  #<START     ; LO BYTE OF IRQ
                                                          ROUTINE
8002: 8D FE 03         18              STA  IRQ         ; PUT IN LO BYTE OF
                                                          VECTOR
8005: A9 80            19              LDA  #>START     ; HI BYTE OF IRQ
                                                          ROUTINE
8007: 8D FF 03         20              STA  IRQ+1       ; PUT IN HI BYTE OF
                                                          VECTOR
                       21
                       22    * SET THE BALL DELTA, TIMER,
                       23    * BALL POSITION
                       24
800A: 20 58 FC         25              JSR  HOME        ; CLEAR SCREEN AND
                                                          HOME CURSOR
800D: A9 FF            26              LDA  #$FF        ; #$FF = -1 IN 2'S
                                                          COMPLEMENT MATH
800F: 8D 48 80         27              STA  XD          ; SET X VELOCITY TO
                                                          -1 UNIT
8012: 8D 49 80         28              STA  YD          ; SET Y VELOCITY TO
                                                          -1 UNIT
8015: A9 14            29              LDA  #20         ; # OF INTS PER
                                                          CYCLE
8017: 8D 44 80         30              STA  RATE        ; SET 'RATE' VALUE
801A: 8D 45 80         31              STA  TIME        ; SET TIME = RATE
801D: A9 14            32              LDA  #20         ; STARTING X,Y POSN
                                                          OF BALL
801F: 8D 46 80         33              STA  XPOS        ; X POSITION OF BALL
                                                          = 20
8022: 8D 47 80         34              STA  YPOS        ; Y POSITION OF BALL
                                                          = 20
                       35
8025: A9 20            36    DRAW1     LDA  #$20        ; INVERSE BLOCK
8027: 8D 4A 80         37              STA  NEWVAL      ; CHARACTER TO PLOT
```

```
802A: AD 46 80  38              LDA  XPOS       ; NEW X POSITION OF
                                                  BALL
802D: 8D 4D 80  39              STA  XP         ; PUT IN XP FOR PLOT
                                                  ROUTINE
8030: AD 47 80  40              LDA  YPOS       ; NEW Y POSITION OF
                                                  BALL
8033: 8D 4E 80  41              STA  YP         ; PUT IN YP FOR PLOT
                                                  ROUTINE
8036: 20 23 81  42              JSR  PLOT       ; DRAW THE BALL
                                                  IMAGE
8039: AD 4C 80  43              LDA  OLDVAL     ; GET VALUE OF
                                                  BACKGROUND CHAR
                                                  'UNDER' BALL
803C: 8D 4B 80  44              STA  SV         ; SAVE IN 'SV'
                                                  LOCATION
                45
                46  * ENABLE INTS. AND ACTIVATE CARD
                47
803F: 58        48              CLI             ; ENABLE INTERRUPTS
8040: AD C0 C0  49              LDA  $C0C0      ; $C0(8+N)0 = TURN
                                                  ON CARD
8043: 60        50              RTS             ; DONE WITH STARTUP!
                51
                52  * VARIABLES
                53
8044: 00        54  RATE    HEX  00         ; # OF INTS PER
                                                  SCREEN CYCLE
8045: 00        55  TIME    HEX  00         ; CURRENT INT #
                                                  COUNTER
8046: 00        56  XPOS    HEX  00         ; X POSN ON SCREEN
                                                  OF BALL (0-39)
8047: 00        57  YPOS    HEX  00         ; Y POSN ON SCREEN
                                                  OF BALL (0-23)
8048: 00        58  XD      HEX  00         ; X VELOCITY OF
                                                  BALL:
                59  *                       ; $01 OR $FF = +1 OR
                                                  -1
8049: 00        60  YD      HEX  00         ; Y VELOCITY OF
                                                  BALL: $01 OR $FF
804A: 00        61  NEWVAL  HEX  00         ; CHARACTER TO
                                                  'PLOT'
804B: 00        62  SV      HEX  00         ; SCREEN VALUE
                                                  'UNDER' THE BALL
804C: 00        63  OLDVAL  HEX  00         ; CHARACTER SAVED BY
                                                  'PLOT'
804D: 00        64  XP      HEX  00         ; X POSN USED BY
                                                  PLOT
804E: 00        65  YP      HEX  00         ; Y POSN USED BY
                                                  PLOT
                66
804F: 00        67  SAVREGS HEX  00         ; ACC SAVING
```

```
                                                LOCATION
8050: 00          68            HEX  00         ; X-REG SAVING
                                                  LOCATION
8051: 00          69            HEX  00         ; Y-REG SAVING
                                                  LOCATION
                  70
                  71  * DISABLE INTS. AND SAVE REGS
                  72
8052: 8D 4F 80    73  START     STA  SAVREGS    ; SAVE ACCUMULATOR
8055: 8E 50 80    74            STX  SAVREGS+1  ; SAVE X REGISTER
8058: 8C 51 80    75            STY  SAVREGS+2  ; SAVE Y REGISTER
                  76
                  77  * DEC TIMER. TIMER TIMED OUT?
                  78  * IF NOT, RETURN. IF SO, RESET
                  79  * TIMER & CONT.
                  80
805B: CE 45 80    81            DEC  TIME       ; TIME = TIME - 1
805E: D0 71       82            BNE  RETURN     ; BACK TO FOREGROUND
                                                  IF NOT DONE
8060: AD 44 80    83            LDA  RATE       ; GET 'RATE' VALUE
8063: 8D 45 80    84            STA  TIME       ; RESTORE TIME
                                                  COUNTER TO RATE
                                                  VAL
                  85
                  86  * RESTORE BACKRND TO SCREEN LOC.
                  87
8066: AD 4B 80    88            LDA  SV         ; GET OLD BACKGROUND
                                                  CHARACTER
8069: 8D 4A 80    89            STA  NEWVAL     ; PUT IN CHARACTER
                                                  TO 'PLOT'
806C: AD 46 80    90            LDA  XPOS       ; CURRENT X POSITION
                                                  OF BALL
806F: 8D 4D 80    91            STA  XP         ; SET XP FOR PLOT
                                                  ROUTINE
8072: AD 47 80    92            LDA  YPOS       ; CURRENT Y POSITION
                                                  OF BALL
8075: 8D 4E 80    93            STA  YP         ; SET YP FOR PLOT
                                                  ROUTINE
8078: 20 23 81    94            JSR  PLOT       ; ERASE OLD BALL
                  95
                  96  * ADD DELTAS TO BALL POSITION
                  97  * CHECK FOR BOUNDARIES. IF HIT,
                  98  * REVERSE DIRECTION.
                  99
807B: AD 46 80   100  NEWX      LDA  XPOS       ; GET CURRENT X POSN
                                                  OF BALL
807E: 18         101            CLC             ; CLEAR CARRY BIT
                                                  FOR AN ADD
807F: 6D 48 80   102            ADC  XD         ; ADD X VELOCITY
                                                  VALUE
```

```
8082: 30 0A     103            BMI  RVRSX     ; IF AT LEFT SIDE OF
                                               SCREEN (X<0)
8084: C9 27     104            CMP  #39       ; AT RIGHT SIDE OF
                                               SCREEN?
8086: F0 06     105            BEQ  RVRSX     ; YES
8088: 8D 46 80  106            STA  XPOS      ; STORE NEW X
                                               POSITION OF BALL
808B: 4C 99 80  107            JMP  NEWY      ; CALCULATE NEW Y
                                               POSITION
808E: AD 48 80  108  RVRSX     LDA  XD        ; GET CURRENT X
                                               VELOCITY
8091: 49 FE     109            EOR  #$FE      ; XD = XD * -1 TO
                                               REVERSE MOTION
8093: 8D 48 80  110            STA  XD        ; STORE NEW X
                                               VELOCITY
8096: 20 15 81  111            JSR  SOUND     ; MAKE A 'BOUNCE'
                                               NOISE
                112
8099: AD 47 80  113  NEWY      LDA  YPOS      ; GET CURRENT Y
                                               POSITION OF BALL
809C: 18        114            CLC            ; CLEAR CARRY BIT
                                               FOR AN ADD
809D: 6D 49 80  115            ADC  YD        ; ADD Y VELOCITY
80A0: 30 0A     116            BMI  RVRSY     ; IF AT TOP OF
                                               SCREEN (Y<0)
80A2: C9 17     117            CMP  #23       ; AT BOTTOM OF
                                               SCREEN (Y=23?)
80A4: F0 06     118            BEQ  RVRSY     ; YES
80A6: 8D 47 80  119            STA  YPOS      ; STORE NEW Y
                                               POSITION OF BALL
80A9: 4C B7 80  120            JMP  DRAW      ; DRAW THE NEW BALL
                                               IMAGE
                121
80AC: AD 49 80  122  RVRSY     LDA  YD        ; GET CURRENT Y
                                               VELOCITY
80AF: 49 FE     123            EOR  #$FE      ; YD = YD * -1 TO
                                               REVERSE MOTION
80B1: 8D 49 80  124            STA  YD        ; STORE NEW Y
                                               VELOCITY
80B4: 20 15 81  125            JSR  SOUND     ; MAKE A 'BOUNCE'
                                               NOISE
                126
                127  * DRAW BALL AT XP,YP AND SAVE
                128  * THE BACKROUND IN SV
                129
80B7: A9 20     130  DRAW      LDA  #$20      ; INVERSE BLOCK
80B9: 8D 4A 80  131            STA  NEWVAL    ; CHARACTER TO PLOT
80BC: AD 46 80  132            LDA  XPOS      ; NEW X POSITION OF
                                               BALL
80BF: 8D 4D 80  133            STA  XP        ; PUT IN XP FOR PLOT
                                               ROUTINE
```

```
80C2: AD 47 80  134          LDA  YPOS       ; NEW Y POSITION OF
                                               BALL
80C5: 8D 4E 80  135          STA  YP         ; PUT IN YP FOR PLOT
                                               ROUTINE
80C8: 20 23 81  136          JSR  PLOT       ; DRAW THE BALL
                                               IMAGE
80CB: AD 4C 80  137          LDA  OLDVAL     ; GET VALUE OF
                                               BACKGRND CHAR
                                               'UNDER' BALL
80CE: 8D 4B 80  138          STA  SV         ; SAVE IN 'SV'
                                               LOCATION
                139
                140 * RESTORE REGS AND RETURN
                141
80D1: AE 50 80  142 RETURN   LDX  SAVREGS+1  ; RESTORE X REGISTER
80D4: AC 51 80  143          LDY  SAVREGS+2  ; RESTORE Y REGISTER
80D7: AD 40 FA  144          LDA  MONIRQ     ; CHECK MONITOR ROM
80DA: C9 85     145          CMP  #$85       ; DBLTM ROM?
80DC: D0 03     146          BNE  RETURN1    ; YES
80DE: A5 45     147          LDA  $45        ; RESTORE ACC FROM
                                               LOCATION $45
80E0: 40        148          RTI             ; RETURN FROM THE
                                               INTERRUPT
80E1: AD 4F 80  149 RETURN1  LDA  SAVREGS    ; RESTORE ACC FROM
                                               SAVREGS
80E4: 40        150          RTI             ; RETURN FROM THE
                                               INTERRUPT
                151
                152 ** ADDRESSES OF 1ST BYTE OF EACH SCREEN LINE
                153
80E5: 00 80 00  154 ADDRLO   HEX  0080008000
80E8: 80 00
80EA: 80 00 80  155          HEX  80008028A8
80ED: 28 A8
80EF: 28 A8 28  156          HEX  28A828A828
80F2: A8 28
80F4: A8 50 D0  157          HEX  A850D050D0
80F7: 50 D0
80F9: 50 D0 50  158          HEX  50D050D0
80FC: D0
80FD: 04 04 05  159 ADDRHI   HEX  0404050506
8100: 05 06
8102: 06 07 07  160          HEX  0607070404
8105: 04 04
8107: 05 05 06  161          HEX  0505060607
810A: 06 07
810C: 07 04 04  162          HEX  0704040505
810F: 05 05
8111: 06 06 07  163          HEX  06060707
8114: 07
```

```
                        164
8115: A2 0A             165  SOUND    LDX  #10          ; VALUE FOR DURATION
                                                          OF TONE
8117: A0 32             166  DLOOP    LDY  #50          ; VALUE FOR PITCH OF
                                                          TONE
8119: AD 30 C0          167           LDA  SPKR         ; CLICK THE SPEAKER
                                                          ONCE
811C: 88                168  PLOOP    DEY               ; Y = Y - 1
811D: D0 FD             169           BNE  PLOOP        ; PITCH LOOP
811F: CA                170           DEX               ; X = X - 1
8120: D0 F5             171           BNE  DLOOP        ; DURATION LOOP
8122: 60                172           RTS               ; DONE WITH THE
                                                          BOUNCE NOISE
                        173
                        174 ** PTR PLOT SAVING BACKRND
                        175
8123: AC 4E 80          176  PLOT     LDY YP            ; GET Y COORDINATE
                                                          TO PLOT AT
8126: B9 E5 80          177           LDA  ADDRLO,Y     ; GET LO BYTE OF
                                                          ADDR FROM TABLE
8129: 85 06             178           STA  PTR          ; PTR = LO BYTE
812B: B9 FD 80          179           LDA  ADDRHI,Y     ; GET HI BYTE FROM
                                                          TABLE
812E: 85 07             180           STA  PTR+1        ; PTR+1 = HI BYTE
8130: AC 4D 80          181           LDY  XP           ; GET X COORDINATE
                                                          TO PLOT AT
8133: B1 06             182           LDA  (PTR),Y      ; GET CHARACTER
                                                          THERE NOW
8135: 8D 4C 80          183           STA  OLDVAL       ; SAVE IN 'OLDVAL'
8138: AD 4A 80          184           LDA  NEWVAL       ; SET ACC TO VALUE
                                                          TO PLOT
813B: 91 06             185           STA  (PTR),Y      ; PUT IT ON THE
                                                          SCREEN
813D: 60                186           RTS               ; DONE WITH THE PLOT
```

INTERRUPT SUMMARY CARD

To turn on interrupts:

1) Set up the vector at $3FE,3FF to point to your interrupt routine.

2) Enable interrupts with the CLI instruction.

3) Turn on the Interrupt Source Card by accessing a memory location in the range $C0(8+N)0 to $C0(8+N)F. ($C0C0 for Slot #4 for example).

Method #2:

If you have used the Workbench, or have BLOADed the IRQON.TB routine, just call it with the syntax:

&"IRQON",ADDR or CALL LOC, ADDR

Where LOC is the address of the IRQON.TB routine and ADDR is the address of the interrupt routine. If the interrupt vector is already set up, interrupts can be re-enabled by omitting the comma and the address variable from the command.

To turn off interrupts:

1) Execute the SEI instruction from machine language.

2) Turn off the Interrupt Source Card by accessing any location in the range $C(N)00 to $C(N)FF. ($C400 for example for slot #4).

Method #2:

If you have the Workbench, or have BLOADed the IRQOFF.TB routine, just call it with the syntax:

&"IRQOFF" or CALL LOC

Where LOC is the address of the IRQOFF.TB routine.